Fault-Tolerant Sorting Networks

by

Yuan Ma

Sc.B. Mathematics
University of Science and Technology of China (1988)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1994

Author ....[illegible]..
Department of Mathematics
June 9, 1994

Certified by ....[illegible]..
Tom Leighton
Professor of Applied Mathematics
Thesis Supervisor

Accepted by ....[illegible]..
David Vogan
Chairman, Departmental Graduate Committee

Fault-Tolerant Sorting Networks

by

Yuan Ma

Submitted to the Department of Mathematics
on June 9, 1994, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis studies sorting circuits, networks, and PRAM algorithms that are tolerant to faults. We consider both worst-case and random fault models, although we mainly focus on the more challenging problem of random faults. In the random fault model, the circuit, network, or algorithm is required to sort all n-input permutations with probability at least $1 - \frac{1}{n}$ even if the result of each comparison is independently faulty with probability upper bounded by a fixed constant. In particular,

- we construct a passive-fault-tolerant sorting circuit with $O(n \log n \log \log n)$ comparators, thereby answering an open question posed by Yao and Yao in 1985,
- we construct a reversal-fault-tolerant sorting network with $O(n \log^{\log_2 3} n)$ comparators, thereby answering an open question posed by Assaf and Upfal in 1990,
- we design an optimal $O(\log n)$-step $O(n)$-processor deterministic EREW PRAM fault-tolerant sorting algorithm, thereby answering an open question posed by Feige, Peleg, Raghavan, and Upfal in 1990, and
- we prove a tight lower bound of $\Omega(n \log^2 n)$ on the number of comparators needed for any destructive-fault-tolerant sorting or merging network, thereby answering an open question posed by Assaf and Upfal in 1990.

All the upper bound results are based on a new analysis of the AKS sorting circuit, which is of interest in its own right. Previously, the AKS sorting circuit was not believed to be fault-tolerant because the expansion properties that were believed to be crucial for the performance of the circuit are destroyed by random faults. The new analysis of the AKS sorting circuit uses a much weaker notion of expansion that can be preserved in the presence of faults. All previous fault-tolerant sorting circuits, networks, and parallel algorithms used $\Theta(\log^2 n)$ depth and/or $\Theta(n \log^2 n)$ comparisons to sort n numbers, and no nontrivial lower bounds were known.

Finally, we use simulation methods to construct practical circuits of small depth that sort most permutations. Simulation results show that such circuits have depth smaller than Batcher's classic circuits and that they are tolerant to a large number of faults.

Thesis Supervisor: Tom Leighton

Title: Professor of Applied Mathematics

Acknowledgments

I am deeply grateful to my advisor, Tom Leighton. He has been an amazing advisor in so many ways that I would not dare to list them all. I particularly thank him for realizing my potential much earlier than I could realize it myself, and I thank him for being such a great guy. All of the results in this thesis are joint work with him.

I would like to thank Greg Plaxton. His beautiful work with Tom attracted me to start my research in the field of computer science. He has been a second advisor and a great friend. The results in Chapters 3 and 5 are joint work with him and Tom.

I would like to thank Dan Kleitman and Charles Leiserson, whose influence on me is far beyond their serving on my thesis committee. I would also like to thank the faculty and my fellow students in the Department of Mathematics and the Laboratory for Computer Science for making MIT such a productive environment. Special thanks go to Professors Hung Cheng, Gian-Carlo Rota, and Richard Stanley.

Over the years, many people have contributed to my education. Without their efforts, I could not have achieved what I have now. Among these people, I would like to recognize the efforts of my uncle Qingzhong Ma, who inspired my interest in mathematics in my childhood, my secondary school teacher Shuoxian Liu, and Professor Keqin Feng in the University of Science and Technology of China.

Finally, I thank my parents and brother for their lifetime support and understanding, and I thank my wife Yiqun Yin, who is my most important discovery during my stay at MIT, for making my life enjoyable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Previous Work

An n-input *sorting circuit* is a circuit that sorts any n input values. It consists of n registers and a collection of comparators. Each *register* holds one of the items to be sorted, and each *comparator* is a 2-input, 2-output device that outputs the two input items in sorted order. The comparators are partioned into *levels* so that each register is involved in at most one comparison in each level. The *depth* of a circuit is defined to be the number of levels in the circuit, and the *size* of a circuit is defined to be the number of comparators in the circuit. For example, a 4-input sorting circuit with depth 5 and size 6 is shown in Figure 1-1.

The study of sorting circuits has intrigued and challenged computer scientists for decades. They have also proved to be very useful for a variety of applications, including circuit switching and packet routing [11]. In the past several years, issues involving the fault-tolerance properties of sorting circuits have gained in importance and attention [4, 6, 23, 24, 26]. In this thesis, we study the problem of constructing sorting circuits that are tolerant to a potentially large number of faults.

The study of fault-tolerant sorting circuits was initiated by Yao and Yao [26] in 1985. In particular, Yao and Yao proposed a fault model in which a faulty comparator simply outputs its two inputs without comparison (i.e., the items are output in the

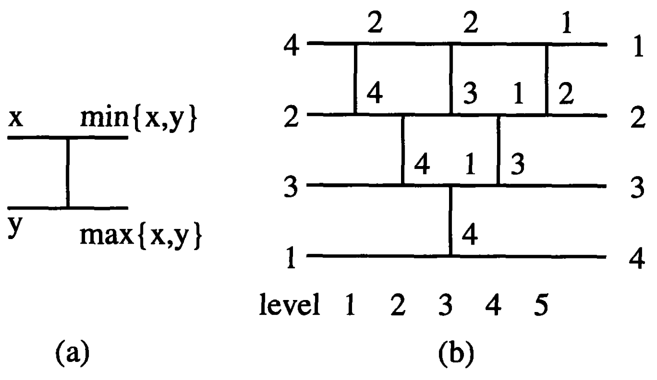

Figure 1-1: (a) A comparator. (b) A sorting circuit.

same order as they are input); we will refer to such a fault as a *passive fault.* They defined an n-input *fault-tolerant* sorting circuit to be a circuit that remains a sorting circuit with probability at least $1 - \frac{1}{n}$ even if each comparator is independently faulty with probability upper bounded by a constant strictly less than 1. They observed that any sorting circuit can be made into a passive-fault-tolerant sorting circuit if each of the original comparators is replicated $O(\log n)$ times.[1] This immediately yields a passive-fault-tolerant sorting circuit with $O(\log^2 n)$ depth and $O(n \log^2 n)$ size. Whether or not there is an alternative approach to fault-tolerance that requires fewer comparators has remained an interesting open question for several years. In particular, Yao and Yao conjectured that $\omega(n \log n)$ comparators are needed to construct a fault-tolerant sorting or merging circuit, but no proof of this conjecture has yet been discovered. (An n-input *merging* circuit is a circuit that merges any pair of $\frac{n}{2}$-item sorted lists to form a sorted list of n items.)

Since Yao and Yao, many researchers have studied fault-tolerant circuits, networks, and algorithms for sorting-related problems in various models. (See [4, 6, 7, 23, 24].) Despite all of these efforts, the $O(\log n)$-gap between the trivial upper and lower bounds has remained open for Yao and Yao's question for both sorting and merging. One approach to narrowing the $O(\log n)$-gap was investigated by Leighton,

[1]In this thesis, all logarithms are taken base 2 unless specified otherwise.

Ma, and Plaxton [15], who constructed an $O(n \log n \log\log n)$ size circuit that sorts any permutation with probability at least $1 - \frac{1}{n}$. However, this circuit does not yield an answer to Yao and Yao's question because sorting any input permutation with high probability is not sufficient to guarantee sorting all input permutations with high probability, and hence it is not sufficient to guarantee that a faulty version of the circuit will be a sorting circuit with high probability. In other words, for a randomly generated fault pattern, there are likely to be some input permutations for which the circuit of [15] fails to sort. (Formally, a *fault pattern* completely specifies which comparators, if any, are faulty and how they fail. That is, a fault pattern of a circuit contains all the information needed to specify the functionality of all the comparators in the circuit.)

Since 1985, several other fault models have also been formulated for the study of fault-tolerant sorting circuits [4, 15]. In the *reversal fault* model, a faulty comparator outputs the two inputs in reversed order regardless of their input order. In the *destructive* fault model, a faulty comparator can output the two inputs in reversed order, and it can also lose one of its two input values and output the other input value in both of the output registers (i.e., the result of a comparison between x and y can be (f, g) where f, g can be any of $\{x, y, \min(x, y), \max(x, y)\}$). In order to tolerate destructive and/or reversal faults, Assaf and Upfal [4] introduced a new computational model for the study of the sorting problem. In their new model, more than n registers are allowed to sort n items and *replicators* are used to copy the item stored in one register to another register. We will call this model the *sorting network model* to distinguish it from the classic *sorting circuit model* in which only n registers are used to sort n inputs and no replicators are allowed. For example, a 2-input sorting network that is tolerant to any single reversal or destructive fault is illustrated in Figure 1-2.

In [4], Assaf and Upfal described a general method for converting any sorting circuit into a reversal-fault-tolerant or destructive-fault-tolerant sorting network. In particular, given an n-input sorting circuit with depth d and size s, the fault-tolerant network produced by the Assaf-Upfal transformation has depth $O(d)$ and size $O(s \log n)$.

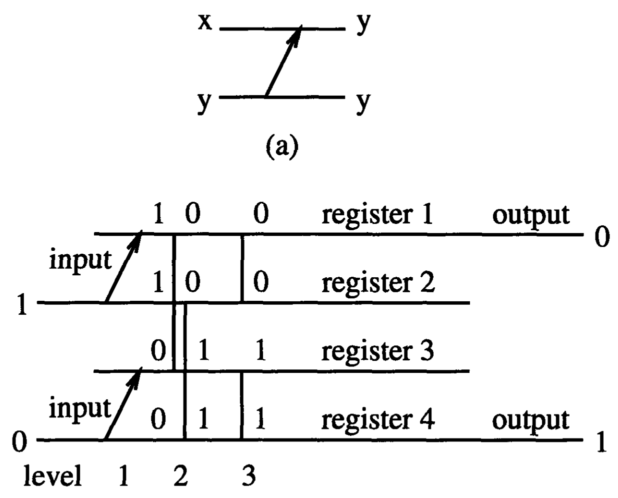

Figure 1-2: (a) A replicator. (b) A 2-input sorting network that tolerates any single reversal or destructive fault.

(Asymptotically, it makes no difference whether or not the replicators are counted towards the size since an optimal network would make a copy of an item only if the item were to be input to a comparator.) When used in conjunction with the AKS sorting circuit [1], this provides a reversal-fault-tolerant or destructive-fault-tolerant sorting network with $O(n \log^2 n)$ size.[2]

The Assaf-Upfal method proceeds by making $\Theta(\log n)$ copies of each item and replacing each comparator with $\Theta(\log n)$ comparators, followed by a majority-enhancing device that is constructed from expanders. As a consequence, the size of the resulting network is increased by a $\Theta(\log n)$ factor. Whether or not there is an alternative approach to reversal-fault-tolerance or destructive-fault-tolerance that can avoid the $\Theta(\log n)$ factor blowup in size (even for the much simpler problem of merging) was an interesting open question posed in [4].

The problem of sorting with faults has also been studied in the PRAM model

[2]In this thesis, as in [4], we will assume that the replicators are fault-free. This is not a particularly unreasonable assumption since replicators can be hard-wired and they do not contain any logic elements. In fact, some of our results can be extended to handle a model in which replicators are also allowed to fail.

of computation. In the PRAM model, it is assumed that faults only occur as incorrect answers to comparison queries and that no item is lost in any comparison. In particular, Feige, Peleg, Raghavan, and Upfal [7] designed a randomized fault-tolerant sorting algorithm that uses $O(\log n)$ expected time on an $O(n)$-processor CREW PRAM. They left open the question of whether or not there is a deterministic fault-tolerant sorting algorithm that runs in $o(\log^2 n)$ steps on $O(n)$ processors.

1.2 Main Results

In this thesis, we develop a passive-fault-tolerant sorting circuit, a reversal-fault-tolerant sorting network, and a fault-tolerant EREW PRAM sorting algorithm that beat the $\Theta(n \log^2 n)$ barrier. We also prove a tight lower bound of $\Omega(n \log^2 n)$ on the size of any destructive-fault-tolerant sorting network. These results partially or wholly resolve the questions posed by Yao and Yao [26], Assaf and Upfal [4], and Feige et al. [7]. In particular,

(i) we construct a passive-fault-tolerant sorting circuit with $O(\log n \log\log n)$ depth and $O(n \log n \log\log n)$ size, which resolves the question posed by Yao and Yao [26] to within an $O(\log\log n)$ factor,

(ii) we construct a reversal-fault-tolerant sorting network with size $O(n \log^{\log_2 3} n)$, which partially resolves the question of Assaf and Upfal [4] for reversal faults,

(iii) we design a fault-tolerant sorting algorithm that runs in $O(\log n)$ steps on an $O(n)$-processor EREW PRAM, which resolves the question of Feige, et al. [7], and

(iv) we prove a tight lower bound of $\Omega(n \log^2 n)$ on the size of any destructive-fault-tolerant sorting or merging network, which answers the question of Assaf and Upfal [4] for destructive faults.

All of the upper bound results are based on some surprisingly strong fault-tolerance properties of the AKS circuit [2]. These upper bound results are surprising because the AKS circuit was not previously believed to be fault-tolerant. In particular, when a constant fraction of the comparators in the AKS circuit fail to work, the expansion property, which plays a central role in the functionality of the fault-free AKS sorting circuit, is lost and it appears that the AKS circuit cannot sort at all. (This is perhaps the main reason that people were unable to make progress on Yao and Yao's question.) The novelty of our work is to show that some loose expansion properties, which can be preserved even in the presence of faults, are sufficient to approximate-sort, and that approximate-sorting can be combined with other methods to sort.

Although we mainly focus on the study of circuits, networks, and algorithms that are tolerant to random faults, all of our techniques apply to worst-case faults. Our results for worst-case faults include the first asymptotically nontrivial upper bound on the depth of worst-case passive-fault-tolerant sorting circuits, a worst-case reversal-fault-tolerant sorting network, and an optimal worst-case fault-tolerant EREW PRAM algorithm for sorting. The techniques in the thesis can also be applied to other sorting-related problems. For example, we will construct a passive-fault-tolerant selection circuit with the asymptotically optimal size of $O(n \log n)$. (Throughout the thesis, a *selection* circuit is defined to be a circuit that outputs the median of its n inputs into a prespecified output register.[3])

1.3 Some Remarks on the Models

Throughout the thesis, we use n to denote the number of input items and $\rho < 1$ to denote the upper bound on the failure probability for each comparator, unless otherwise specified. A circuit, network, or algorithm for solving a problem Q is defined to be (ρ, ϵ)-*fault-tolerant* if it satisfies the following condition: when each comparator or comparison is independently faulty with probability upper bounded by ρ, with

[3]Some researchers define a selection circuit to be a circuit that outputs the small half of its input values to a prespecified set of output registers and outputs the large half of its input values to the other registers. In the fault-free case, the asymptotic complexity of a selection circuit is independent of which definition we use, according to Exercise 17 on page 239 of [10]. It turns out that our results on selection circuits with passive faults also hold independent of the choice of definition.

probability at least $1-\epsilon$, a faulty version of the circuit, network, or algorithm remains a circuit, network, or algorithm that solves Q on all possible input instances. When we simply refer to a circuit, network, or algorithm as *fault-tolerant*, we mean that the circuit, network, or algorithm is $(\rho, \frac{1}{n})$-fault-tolerant for some constant $\rho < 1$. For any constant c, all of our constructions can be made into constructions with success probability at least $1 - \frac{1}{n^c}$ with no more than a constant factor increase in size, but we will be content to demonstrate success probability at least $1 - \frac{1}{n}$ in most parts of the thesis.

As an alternative, we could have defined the notion of (ρ, ϵ)-fault-tolerance by assuming that ρ is exactly the failure probability of each comparator or comparison, as opposed to an upper bound on such probability. In general, these two definitions are not equivalent, and it is straightforward to show that any (ρ, ϵ)-fault-tolerant circuit, network, or algorithm defined in the preceding paragraph is also (ρ, ϵ)-fault-tolerant according to the alternative definition.[4] Hence, to obtain the strongest possible results, we will use the definition of the preceding paragraph for all upper bound results, which include all but Theorems 3.2.1, 3.2.2, and 4.2.1, and we will use the alternative definition for all lower-bound results, which include Theorems 3.2.1, 3.2.2, and 4.2.1.

Finally, we point out that all of the comparators used in the constructions of our circuits and networks move the small input to its "top" register and the large input to its "bottom" register. Following the notation of Knuth [10], this means that all of our circuits and networks are *standard*. All of our lower bounds are proved for the general case, i.e., we do not assume that the circuit is standard in the lower bound proofs. In the fault-free case, it has been proved that any non-standard sorting circuit can be converted into a standard sorting circuit with the same depth and size (see Exercise 16 on page 239 of [10]). However, we do not know if a similar result is true when the circuit is subject to passive faults.

[4]A detailed discussion of this phenomena in the context of Boolean circuits with noisy gates can be found in [21].

1.4 Organization of the Thesis

The rest of the thesis is organized into chapters as follows. In Chapter 2, we prove that the AKS circuit has certain useful fault-tolerance properties. In Chapter 3, we use the AKS circuit to construct fault-tolerant sorting circuits, networks, and EREW algorithms. In Chapter 4, we prove a lower bound on the size of any destructive-fault-tolerant sorting network. In Chapter 5, we present some simulation results on constructing small depth circuits that, either with or without faults, sort most input permutations. We conclude in Chapter 6 with discussions on some future research directions.

We remark that preliminary versions of the results in Chapters 2, 3, and 4 have appeared in [13, 14, 15].

Chapter 2

The Analysis of a Modified AKS Circuit

In this chapter, we show that the AKS circuit [2] has certain fault-tolerance properties under both the passive and reversal fault models. These fault-tolerance properties will be the cornerstone for most of the fault-tolerant sorting circuits, networks, and algorithms in Chapter 3. We believe that our new analysis for the AKS circuit is of separate interest in its own right.

The chapter is divided into three sections: Section 2.1 explains why the previously known analyses of the AKS circuit are not sufficient to establish the desired fault-tolerance properties and highlights the major difficulties in the new analysis. Section 2.2 contains a brief description of the AKS circuit and the relevant parameter choices. Section 2.3 proves the main theorem of the chapter and its corollary.

2.1 The Need for a New Analysis of the AKS Circuit

The key component of the AKS circuit is the *ε-halver*. An m-input circuit is called an ε-halver if, on any m distinct inputs, it outputs at most εk of the k smallest (largest) inputs into the bottom (top) $\frac{m}{2}$ registers for any $k \leq \frac{m}{2}$. For any constant $\varepsilon > 0$, a bounded-depth ε-halver can be built from an expander as follows. Take an $\frac{m}{2} \times \frac{m}{2}$ d-regular $(\varepsilon, \frac{1-\varepsilon}{\varepsilon})$-bipartite expander with vertex sets A and B, where d is a constant dependent on ε. (A bipartite $\frac{m}{2} \times \frac{m}{2}$ graph with vertex sets A and B is called an (α, β)-*expander* if any $k \leq \alpha m$ nodes in A (or B) are connected to at least βk nodes in B (or A). Explicit constructions of expanders can be found in [18].) Assign each vertex in A to a register in the top half of the circuit, and assign each vertex in B to a register in the bottom half of the circuit. Partition the edges of the expander into d disjoint matchings. Assign a comparator between two registers at level i in the halver if and only if the corresponding vertices are connected by an edge in the ith matching of the expander.

To see why this construction yields an ε-halver, we assume for the purposes of contradiction that the circuit is not an ε-halver. Without loss of generality, we can assume that there exist m distinct inputs and an integer $k \leq \frac{m}{2}$ such that strictly more than εk of the k smallest inputs are output into R, a set of strictly more than εk registers in the bottom half of the circuit. Let R' be the set of registers in the top half of the circuit that are connected to some registers in R. It is not hard to show that all registers in R' contain outputs with rank at most k. Therefore, all the $|R| + |R'| \geq |R| + \frac{1-\varepsilon}{\varepsilon}\varepsilon k = |R| + (1-\varepsilon)k > k$ output items contained in either R or R' have rank at most k. This is a contradiction.

It would be nice if the ε-halver could tolerate random faults automatically or if the ε-halver could be made fault-tolerant with a $o(\log n)$ factor increase in the depth. (For example, if this were possible, Yao and Yao's question would have an easy answer.) As we have seen in the previous paragraph, however, the fact that $|R'| \geq (1-\varepsilon)k$ is critical to guarantee the ε-halver property, and this fact in turn depends on the expansion property of the expander. Unfortunately, the following observation indicates that the expansion is lost in the presence of faulty comparators and that the cost of achieving fault-tolerant expansion is very high. For example, if $d = \Theta(1)$ and each comparator in the ε-halver constructed above is independently faulty with constant probability, then with high probability there exists a set of $k = \Theta(m)$ registers in the bottom half of the circuit for which all associated comparators are faulty. Hence, if the k smallest inputs are all input to these registers, then the inputs cannot be moved to the top half of the circuit. This shows that the ε-halver itself cannot withstand random faults. Moreover, even if we increase the depth of an ε-halver by a nonconstant $o(\log n)$ factor, any constant number of registers are connected to only $o(\log n)$ comparators, and, with probability $\omega(\frac{1}{n})$, these registers are not connected to any working comparators. (Using more careful arguments, we can actually show that with probability approaching 1, there exists a set of $\omega(1)$ registers that are not connected to any working comparators.) Hence, if a constant number of the smallest inputs are input into these registers, they cannot be reliably moved to the top half of the halver.

Since expansion plays a central role for both the functionality and the correctness proof of the AKS circuit, the loss of such expansion in the presence of faulty comparators is a fatal problem for all previously known analyses of the AKS circuit. In fact, the loss of the expansion property even makes the approach of using the AKS circuit to construct fault-tolerant sorting circuits seem to be hopeless. The novelty of our work is to show that, even without guaranteeing "local" expansion at any single expander, it is possible to enforce a certain "global" expansion property that is sufficient to guarantee that the AKS circuit functions reasonably well.

2.2 The Description of a Modified AKS Circuit

In this section, we describe a modified AKS circuit that will be shown to possess certain fault-tolerance properties. Our description consists of two parts. In the first part, we briefly describe a modification of the AKS circuit described by Paterson in [20]. We modify several parameter choices, and replace the so-called *separators* of [20] by a new family of building blocks that we call *partitioners.* In the second part, we further modify the AKS circuit into an l-AKS circuit, where l is a parameter to be specified later that will correspond to the amount of fault-tolerance attained by the circuit.

We will be content with proving that certain parameter choices guarantee the desired fault-tolerance properties. No attempt will be made to keep the involved constants small. In particular, an extremely large constant (much larger than the previously best known constant for the AKS circuit) is hidden behind the O-notation for all of our circuits, networks, and algorithms.

For simplicity, we will not give a completely detailed description of the modified AKS circuit. Instead, we will follow the description in [20] whenever possible. In particular, we will use the same letters to denote the same quantities as in [20] unless specified otherwise.

We will use the same AKS tree construction as in [20]. In particular, the circuit works in *stages* starting from stage 1, and each stage consists of a constant number of levels. We choose the same parameters associated with the AKS tree as in [20]:

$$A = 3,\ \nu = \frac{43}{48},\ \text{and}\ \lambda = \frac{1}{8}. \tag{2.1}$$

Also as in [20], we choose

$$\mu = \frac{1}{36}. \tag{2.2}$$

Instead of using the parameters ε and δ as in [20], we use parameters ϕ and σ with the relation

$$\phi = 65\left(\frac{4\sigma A}{\nu(1-\frac{1}{2A})}\right)^2. \tag{2.3}$$

In a certain sense, our parameters $\frac{1}{\phi}$ and $\frac{1}{\sigma}$ correspond to the parameters ε and δ in [20]. We do not specify the choices of ϕ and σ here, but we will see in the next section that a sufficiently large ϕ is good for our purposes. For now, we merely assume that $\sigma > 1$. The parameters μ and σ, like the parameters μ and δ in [20], have nothing to do with the description of the circuit and will be used only in the analysis of the circuit.

At each node of the AKS tree, a sorting-related device is applied. Separators were used in [20], and near-sorting circuits were used in [1]. The separator is constructed from a constant number of ε-halvers. Informally, a *separator* is slightly more powerful than a halver in the sense that a separator not only moves most of the inputs to the correct half but also moves most of the "extreme" inputs close to the extreme positions. Since, as discussed in Section 2.1, we cannot build ε-halvers that are both efficient and fault-tolerant, we cannot construct efficient and fault-tolerant separators either.

In [20], ε-halvers and separators are defined in terms of their functionality. The procedure given for building an ε-halver from an expander, and for building a separator from ε-halvers, represents only one of many possible constructions. In this thesis, we will be interested in the specific constructions given in [1] and [20], but these constructions will likely fail to have the properties needed to be ε-halvers or separators once faults are introduced. So, to avoid confusion, we define a *ϕ-divider* with m inputs to be a circuit constructed by using an $\frac{m}{2} \times \frac{m}{2}$ d-regular bipartite $(\frac{1}{\phi+1}, \phi)$-expander to connect the top half and the bottom half of the m registers in the same fashion as we construct the ε-halver in the previous section. An m-input *(λ, ϕ)-partitioner* is constructed by applying dividers in rounds: We first apply an m-input ϕ-divider to all m registers. Then, we apply an $\frac{m}{2}$-input ϕ-divider to the top $\frac{m}{2}$ registers and another $\frac{m}{2}$-input ϕ-divider to the bottom $\frac{m}{2}$ registers. Next, we apply an $\frac{m}{4}$-input ϕ-divider to the top $\frac{m}{4}$ registers and another $\frac{m}{4}$-input ϕ-divider to the bottom $\frac{m}{4}$ registers (we do not do anything to the $\frac{m}{2}$ "middle" registers). We then apply another $\frac{m}{8}$-input ϕ-divider to the top $\frac{m}{8}$ registers and another $\frac{m}{8}$-input ϕ-divider to the bottom $\frac{m}{8}$ registers. We keep doing this until we have applied a divider to a group with at most λm registers. Altogether, we apply the dividers for $1 + \log\lceil\frac{1}{\lambda}\rceil$ rounds. In the proof of Theorem 2.3.1, we will refer to the dividers applied in the ith round of a partitioner as *the ith round dividers*.

Even though the construction of a partitioner (divider) is the same as the separator (halver) construction used in [20], a partitioner (divider) is conceptually different from a separator (halver) in that a separator (halver) is defined based on its input-output behavior and a partitioner (divider) is explicitly constructed from bipartite expanders. Of course, a fault-free partitioner (divider) is one type of separator (halver).

If we were only interested in passive-fault-tolerant sorting circuits, the modified AKS circuit just described would be sufficient. However, to construct reversal-fault-tolerant circuits and networks, we need to further modify the AKS circuit into an l-AKS circuit, as follows.

For any given integer $l > 0$, we use the following general technique to modify any family of circuits $\mathcal{F}$ into another family of circuits $\mathcal{F}'$ with parameter l. In general, an m-input circuit $\mathcal{C}'$ in $\mathcal{F}'$ is constructed from an $\frac{m}{l}$-input circuit $\mathcal{C}$ in $\mathcal{F}$ as follows. For each $i \leq \frac{m}{l}$, replace the ith register in $\mathcal{C}$, r_i, by a group of l registers, $r_{i_1}, \ldots, r_{i_l}$. This group will be referred to as a *block* corresponding to register r_i. Replace each comparator in $\mathcal{C}$ that connects registers r_i and r_j by a $2l$-input and $4l$-depth odd-even transposition circuit that connects $r_{i_1}, \ldots, r_{i_l}$ and $r_{j_1}, \ldots, r_{j_l}$. (The reason for doing this can be found in Lemma 2.3.1 and the proofs of Lemma 2.3.2 and Theorem 2.3.1.) Such a circuit $\mathcal{C}'$ in $\mathcal{F}'$ will be referred to as the circuit constructed from $\mathcal{C}$ in $\mathcal{F}$ by applying $2l$-input and $4l$-depth odd-even transposition circuits. In particular, an m-input l-AKS circuit is constructed from an $\frac{m}{l}$-input modified AKS circuit described earlier by applying $2l$-input and $4l$-depth odd-even transposition circuits. For example, a 1-AKS circuit is the modified AKS circuit described earlier with each of the comparators replicated 4 times. This technique is essential to obtain reversal-fault-tolerance, and will be applied again to construct some other circuits such as the circuits in Lemma 2.3.2.

Assume that an l-AKS circuit $\mathcal{C}'$ is constructed from a modified AKS circuit $\mathcal{C}$ by applying $2l$-input and $4l$-depth odd-even transposition circuits. In the AKS tree for $\mathcal{C}$, each node contains a set of registers R. The *l-AKS tree* for the l-AKS circuit $\mathcal{C}'$ is constructed from the AKS tree for $\mathcal{C}$ with each register $r \in R$ replaced by the block of registers corresponding to r during the construction of $\mathcal{C}'$ from $\mathcal{C}$. The *capacity* of an l-AKS tree node X for $\mathcal{C}'$ is defined to be the maximum number of registers (not the maximum number of blocks) allowed to be contained in X, which is equal to l times the capacity of the corresponding AKS tree node for $\mathcal{C}$.

Finally, we do not run the l-AKS circuit all the way to completion. Instead, we stop an m-input circuit immediately before the first stage where the capacity of the root for the l-AKS tree is strictly less than $\sqrt{m}$. This guarantees that the capacity of any node in the l-AKS tree is at least $\sqrt{m}$. For ease of reference, we call such a circuit a *partial l-AKS circuit.* Since we only need partial l-AKS circuits with $l \leq m^{\frac{1}{8}}$ (see Theorem 2.3.1), we do not have to worry about the case where the capacity of a certain node is too small (this was handled by Batcher's sorting circuit in [20]). We do need to consider integer rounding problems, but this can be easily handled in the same way as in [20], and we will not address this particular problem hereafter.

2.3 The Main Theorem

We start this section with some definitions to be used in the statement and proof of the main theorem. As in [20], we assign each m-input l-AKS tree node a *natural interval* as follows: the natural interval of the root is $[1, m]$; if the natural interval of a node X is $[\alpha, \beta]$, then the natural intervals of the left and right children of X are the left and right halves of $[\alpha, \beta]$, respectively. Intuitively, when a permutation of $\{1, \ldots, m\}$ is input to the l-AKS circuit, the natural interval of a node represents the range of numbers that the registers in the node "should" contain. The following concepts of content, strangeness, and potential are all dependent on which permutation is input to the circuit and which level (time) of the circuit we are interested in, but we do not include the permutation or time as part of the notations since we will only focus on a fixed input permutation in the proof of Theorem 2.3.1 and since the time will be clear from the context whenever we use these concepts. We define $c(r)$, the *content* of a register r at time t, as the input contained in r at time t. We also define $s(r)$, the *strangeness* of $c(r)$ (or of r at time t), to be the number of levels that $c(r)$ needs to move upward in the l-AKS tree from $c(r)$'s current node to the first node whose natural interval contains $c(r)$. Equivalently, we say that $c(r)$ (or r at time t) is $s(r)$-strange. Given any constant $\sigma > 1$, we define the potential of a register r as:

$$P_\sigma(r) = \begin{cases} \sigma^{s(r)-1} & \text{if } s(r) \geq 1 \\ 0 & \text{otherwise.} \end{cases}$$

We define the potential of any set of registers R as:

$$P_\sigma(R) = \sum_{r \in R} P_\sigma(r).$$

In particular, the potential of a node X in the l-AKS tree is

$$P_\sigma(X) = \sum_{r \in X} P_\sigma(r).$$

As far as we know, potential functions have not been used in any previous analysis of the AKS circuit. A similar potential function was used in [16] to prove certain fault-tolerance properties of the multibutterfly circuit for routing. Unfortunately, our use of the potential function here is much more complex than that in [16]. The next theorem provides an upper bound on the number of strange items and as such is analogous to inequality 2 in [20]. Recall that, as discussed in Section 2.2, the capacity of an l-AKS tree node is the number of registers (not the number of blocks) in the node.

Theorem 2.3.1 *Under both the passive and the reversal fault models, for any $l \leq m^{\frac{1}{8}}$, if ϕ is a sufficiently large constant and $\rho > 0$ is less than a sufficiently small constant, then a randomly faulty m-input partial l-AKS circuit satisfies the following inequality with probability at least $1 - \rho^{\Theta(l \log m)}$: For all input permutations and all nodes X in the l-AKS tree,*

$$P_\sigma(X) \leq \mu \operatorname{cap}(X). \tag{2.4}$$

In the theorem, ρ is assumed to be less than a sufficiently small constant, say, ρ_0. Constant ρ_0 and the constant behind the Θ-notation in the theorem are both dependent on ϕ and σ. Most importantly, however, ρ is not necessarily a constant even though ρ is upper bounded by the constant ρ_0, and the constant behind the Θ-notation of the theorem is independent of ρ. It should be mentioned that ρ will be quite small in all of our applications of the theorem and its corollary (see Corollary 2.3.1).

To prove the theorem, we first need to prove a few lemmas. We define a circuit $\mathcal{N}$ (possibly containing faulty comparators) to be a Δ-*approximate-sorting* circuit if, on all the possible input permutations, $\mathcal{N}$ outputs every item to within Δ positions of its correct position.

Lemma 2.3.1 *Under both the passive and the reversal fault models, for any constant $\vartheta > 0$, when ρ is less than a sufficiently small constant (depending on ϑ), a randomly faulty $2l$-input, $4l$-depth odd-even transposition circuit is a ϑl-approximate-sorting circuit with probability at least $1 - \rho^{\Theta(l)}$.*

Proof: We will only present the proof for reversal faults. The same proof is also valid for passive faults. In fact, for passive faults, our proof technique can be used to prove that the odd-even transposition circuit is indeed a $(\rho, \rho^{\Theta(l)})$-passive-fault-tolerant sorting circuit rather than a $(\rho, \rho^{\Theta(l)})$-passive-fault-tolerant ϑl-approximate-sorting circuit. Throughout the proof, we will assume that ρ is less than a sufficiently small constant, depending on ϑ. We do not know if a similar result can be proved for ρ near $\frac{1}{2}$.

Let $\mathcal{C}$ be the odd-even transposition circuit described in Lemma 2.3.1, and let $\mathcal{C}'$ be a randomly generated faulty version of $\mathcal{C}$. By the 0-1 principle, we only need to show that

$$\text{Prob}\,(\exists \text{ 0-1 sequence } s \text{ such that } \mathcal{C}' \text{ does not } \vartheta l\text{-approximate-sort } s) \leq \rho^{\Theta(l)}.$$

Notice that the total number of possible 0-1 input sequences to $\mathcal{C}$ is at most 2^{2l}. Hence, when ρ is less than a sufficiently small constant, to prove the above inequality, we only need to prove that for any fixed 0-1 input sequence s,

$$\text{Prob}(\mathcal{C}' \text{ does not } \vartheta l\text{-approximate-sort } s) \leq \rho^{\Theta(l)}. \qquad (2.5)$$

In what follows, we will prove inequality 2.5 for a fixed s. Assuming that $\mathcal{C}'$ does not ϑl-approximate-sort s, we prove that the behavior of the comparators in $\mathcal{C}'$ satisfies a certain condition that is only satisfied with probability upper bounded by $\rho^{\Theta(l)}$. Without loss of generality, we will assume that on input sequence s, $\mathcal{C}'$ outputs a 0 at least ϑl away from its correct position. (This assumption only affects the probability bound in inequality 2.5 by at most a factor of 2.)

Let k be the number of 0s in sequence s. From the top to the bottom, we label all of the $2l$ registers in $\mathcal{C}$ by $r_1, r_2, \ldots, r_{2l}$. We focus on the positions of the 0s at each level of $\mathcal{C}'$ as the 0s and 1s move forward through $\mathcal{C}'$. As the 0s move forward through $\mathcal{C}'$, they gradually move upward in the 1s. In particular, a 0 in r_i that is correctly compared to a 1 in r_{i-1} at level t will move to r_{i-1}. Intuitively, if most of the comparators involved work correctly, 0s will move upward as they move forward. The problem in analyzing the movement of the 0s, however, is that they can block each other's upward movement. In particular, if one 0 moves the wrong way, it can cause a ripple effect much like a multicar collision on a highway. In the process of generating $\mathcal{C}'$ from $\mathcal{C}$, each of the comparators in $\mathcal{C}$ can be faulty with probability up to ρ. Hence, there are likely to be many such collisions. In addition to slowing things down, such collisions also introduce dependence issues in the analysis of the probabilistic movement of the 0s.

In order to get around these difficulties, we model the moves made by the 0s with a $k \times 4l$ matrix $A = (a_{i,j})$ of random biased coins. In particular, $a_{i,j} = H$ with probability at least $1 - \rho$ and $a_{i,j} = T$ with probability at most ρ. The coin at $a_{i,j}$ will be used to determine whether or not the comparator entered by the ith 0 at level j is faulty. (We number the 0s, starting at 1, from the top to the bottom at the outset, and we never alter the relative order of the 0s in $\mathcal{C}'$.) Note that if two 0s enter the same comparator, the two associated coin flips could conflict in determining whether or not the comparator is faulty. However, we can assume that comparisons between two 0s are resolved according to the initial ordering of the 0s; we do not need to refer to the coin flips in such a case. Note that matrix A completely determines the behavior of $\mathcal{C}'$ on the fixed s.

If at level t the ith 0 is compared to a 1 above, then the 0 moves upward one position if and only if $a_{i,t} = H$. If at level t the ith 0 is compared to a 1 below, then the 0 moves downward if and only if $a_{i,t} = T$. If at level t the ith 0 is compared to a 0 above (i.e., if it is blocked from above by the $(i-1)$th 0), then the ith 0 stays in the same register, and we change the value of $a_{i,t}$ to Z without checking to see whether $a_{i,t} = H$. If at level t the ith 0 is compared to a 0 below (i.e., if it is blocked from below by the $(i+1)$th 0), then the ith 0 stays in the same register and we change the value of $a_{i,t}$ to Z' without checking to see whether $a_{i,t} = H$.

After these modifications, matrix A now contains some Zs and Z's. Call the new matrix $A^* = (a^*_{i,j})$. Note that A^* completely determines the functionality of C' on the fixed s and vice versa. This fact makes it possible for us to prove inequality 2.5 by analyzing A^*.

Define t_k to be the last level where the kth 0 was blocked by the $(k-1)$th 0. In other words, t_k is the maximum integer such that $a^*_{k,t_k} = Z$. Next, define t_{k-1} to be the last level where the $(k-1)$th 0 was blocked by the $(k-2)$th 0 strictly before level t_k. In other words, t_{k-1} is the largest integer such that $t_{k-1} < t_k$ and $a^*_{k-1,t_{k-1}} = Z$. Proceeding in a similar fashion, for $j = k-2, k-3, \cdots, 2$, define t_j to be the largest integer such that $t_j < t_{j+1}$ and $a^*_{j,t_j} = Z$. (It may be that $a^*_{j,t} \neq Z$ for all $t < t_{j+1}$, in which case we set $t_j = t_{j-1} = \cdots = t_1 = 0$.) If $t_2 = 0$, let $t_1 = 0$; if $t_2 > 0$, let t_1 be the largest integer such that $t_1 < t_2$ and the first 0 is located at r_1 immediately before level t_1 (if the first 0 never reaches r_1 strictly before level t_2 then set $t_1 = 0$).

Let S denote the string of coins

$$a_{1,t_1+1}a_{1,t_1+2}\cdots a_{1,t_2-1}a_{2,t_2+1}a_{2,t_2+2}\cdots a_{2,t_3-1}\cdots a_{k,t_k+1}a_{k,t_k+2}\cdots a_{k,4l}.$$

Let n_H denote the number of heads in S and n_T denote the number of tails in S. It is easy to see that S contains $4l - k - t_1 + 1$ coins, which implies that

$$n_T + n_H = 4l - t_1 - k + 1. \tag{2.6}$$

Roughly speaking, the number of upward moves of the kth 0 is given by $n_T - n_H$. However, this bound is not accurate because of *boundary effects* (i.e., caused by the 0s piling up against r_1). To be more precise, we analyze the movement of the 0s by considering two cases.

Case 1: $t_j > 0$ for all j such that $1 \leq j \leq k$. In this case, the first 0 is received at r_1 immediately before level t_1. On the other hand, the total number of downward moves corresponding to S is at most n_T. Hence, at the end of C', the kth 0 is at most n_T positions away from r_k. By our assumption that a 0 is output to at least ϑl away from its correct position, the kth 0 must be output to at least ϑl away from its correct position r_k. Hence,

$$n_T \geq \vartheta l.$$

Case 2: $t_j = 0$ for some j such that $1 \leq j \leq k$. By definition,

$$t_1 = 0. \tag{2.7}$$

In this case, when analyzing the upward moves of 0s corresponding to S, there is no boundary effect to consider. Therefore, the number of upward moves is given by $n_H - n_T$. Since the kth 0 can initially be at most $2l - k$ positions away from r_k, and since the kth 0 is output to at least ϑl away from its correct position r_k, we conclude that

$$2l - k - n_H + n_T \geq \vartheta l.$$

Adding this inequality to equation 2.6 and using equation 2.7, we obtain that

$$n_T \geq \frac{(2+\vartheta)}{2} l \geq \vartheta l,$$

where we have assumed that $\vartheta \leq 2$ since there is nothing to prove for $\vartheta > 2$.

In both Case 1 and Case 2, we have proven that

$$n_T \geq \vartheta l. \tag{2.8}$$

We next show that for a random matrix A, the probability that A^* contains a sequence S such that inequality 2.8 holds is at most $\rho^{\Theta(l)}$.

Let us define $a_{i,j}$ to be *next to $a_{u,v}$ in A* if and only if: (i) i is equal to u or $u+1$, and (ii) j is equal to v or $v+1$. According to the construction of S, the second element of S is next to the first element of S in A, the third element of S is next to the second element of S in A, and so on. Hence, when the location of the ith element of S is given in A, there are at most 3 ways for the $(i+1)$th element in S to be located in A. In addition, the number of ways that the first element of S is located in A is upper bounded by $k \cdot 4l \leq 2l \cdot 4l \leq 8l^2$. On the other hand, $|S| \leq 4l$. Hence, the number of ways for choosing the location of S in A is at most

$$8l^2 \cdot 3^{4l}. \tag{2.9}$$

By a standard Chernoff bound argument, when the location of S in A is given, the probability that inequality 2.8 holds is at most

$$\rho^{\Theta(l)} \tag{2.10}$$

for ρ less than a sufficiently small constant (depending on ϑ). Multiplying the bounds of inequalities 2.9 and 2.10 and setting ρ to less than a sufficiently small constant, we find inequality 2.8 holds with probability at most $\rho^{\Theta(l)}$. This completes the proof of inequality 2.5 as well as the proof of Lemma 2.3.1. ■

In the next lemma, the circuit $\mathcal{N}$ is the parallel union of s disjoint circuits, $\mathcal{N}_1, \ldots, \mathcal{N}_s$. Each $\mathcal{N}_i$ is constructed from a ϕ-divider by replacing each register with a block of l registers and each comparator with a $2l$-input $4l$-depth odd-even transposition circuit. By definition, each of the ϕ-dividers is constructed from a d-regular bipartite expander, and thus has depth d, which is a constant depending in ϕ. Hence, the depth of each $\mathcal{N}_i$ and the depth of $\mathcal{N}$ are $4dl$. A block or a register will be called a *bottom* (*top*) block or register in $\mathcal{N}$ if it is at the bottom (top) half of some $\mathcal{N}_i$. For a set of bottom (top) registers R, we use $N(R)$ to denote the set of top (bottom) registers that are connected to at least one register in R by some odd-even transposition circuit. In the next lemma, n_i denotes the number of inputs to $\mathcal{N}_i$, and n denotes the number of inputs to $\mathcal{N}$.

Lemma 2.3.2 *Let R be a fixed set of bottom (resp., top) registers and b be the number of blocks that contain at least one register in R. Under both the passive and the reversal fault models, when ϕ is large enough, a randomly faulty version of $\mathcal{N}$ has the following* *property with probability at least $1 - \rho^{\Theta(bl+|R|)}$: On all input permutations such that each $\mathcal{N}_i$ contains at most $\frac{49}{100}n_i$ inputs with rank at most $k \le \frac{49}{100}n$ (resp., at least $k \ge \frac{51}{100}n$), if every register in R contains an output with rank at most k (resp., at least k), then at least $\frac{\phi}{4}|R|$ registers in $N(R)$ contain outputs with rank at most k (resp., at least k).*

Note that in the $1 - \rho^{\Theta(bl+|R|)}$ lower bound for the success probability claimed in the lemma, we could omit the $|R|$ term without affecting the meaning of the lemma since $bl \ge |R|$. However, we have chosen to include the term $|R|$ for ease of future applications.

Proof of Lemma 2.3.2: The two claims of the lemma are symmetric, and so we consider only the case where R is a set of bottom registers. We make use of the 0-1 principle. Suppose that each comparator is randomly set to be faulty (with probability ρ or less) ahead of time. We will show that with probability at least $1 - \rho^{\Theta(bl)}$, the resulting circuit has the following property: On all 0-1 input sequences with exactly k 0s such that each $\mathcal{N}_i$ contains at most $\frac{49}{100}n_i$ 0-inputs, if all of the registers in R contain a 0-output, then at least $\frac{\phi}{4}|R|$ registers in $N(R)$ contain a 0-output.

Let

$$\theta = \frac{1}{d\phi}. \tag{2.11}$$

We focus on the bottom blocks containing at least one register in R. We say that such a block is *dense* if it contains strictly more than $d\theta l$ registers in R; we say that such a block is *sparse* if it contains at least one but at most $d\theta l$ registers in R. Note that when $d\theta l < 1$, there exists no sparse block. Let b_{i1} be the number of dense blocks in $\mathcal{N}_i$, and let b_{i2} be the number of sparse blocks in $\mathcal{N}_i$. We call a top (bottom) block B' a *neighboring block* of a bottom (top) block B if there is an odd-even transposition circuit connecting B and B' in $\mathcal{N}$. Note that each block has exactly d neighboring blocks because each of the corresponding ϕ-dividers has depth d. We call a block B *good* if all of the (no more than $d + d^2$) odd-even transposition circuits associated with B or associated with any of the d neighboring blocks of B are θl-approximate-sorting circuits. Let b'_{i1} be the number of good dense blocks in $\mathcal{N}_i$, and b'_{i2} be the number of good sparse blocks in $\mathcal{N}_i$.

When a good block B is connected to a top block B' by a θl-approximate-sorting circuit $\mathcal{M}$, the correctness of the following two simple observations is straightforward:

Observation 1. At most θl 0s can be moved from B' to B through $\mathcal{M}$.

Observation 2. If B contains at least one 0 at the end of $\mathcal{M}$, then B' contains at least $(1-\theta)l$ 0s at the end of $\mathcal{M}$.

The goal of our proof of Lemma 2.3.2 is to find a large number of 0s in $N(R)$ in comparison with the number of 0s in R. This goal will be achieved as follows. From observation 1 above, each good dense block contains 0s throughout the circuit. Therefore, we can hope to use the expansion property of the ϕ-divider to find many 0s in the neighboring blocks of a good dense block. From observation 2, for each good sparse block, its unique neighboring block at the end of $\mathcal{N}$ contains many 0s, compared with the number of 0s contained in the sparse block. In particular, we prove the lemma by considering the following two cases.

Case 1: $\sum_{1\le i\le s} b_{i1} \ge \frac{1}{\phi}\sum_{1\le i\le s} b_{i2}$.

By Lemma 2.3.1, each dense block is good with probability at least $1-(d+d^2)\rho^{\Theta(l)} = 1-\rho^{\Theta(l)}$ provided that ρ is less than a sufficiently small constant (particularly, we can assume ρ to be small compared with $d+d^2$). A standard application of the Chernoff bound now implies that when ρ is less than a sufficiently small constant, the following inequality holds with probability at least $1-\rho^{\Theta(l\sum_{1\le i\le s} b_{i1})}$:

$$\sum_{1\le i\le s} b'_{i1} \ge \frac{2}{3}\sum_{1\le i\le s} b_{i1}. \tag{2.12}$$

By the assumption of Case 1, we have $\sum_{1\le i\le s} b_{i1} \ge \frac{1}{\phi+1}\sum_{1\le i\le s}(b_{i1}+b_{i2}) = \frac{b}{\phi+1}$. Hence, inequality 2.12 holds with probability at least $1-\rho^{\Theta(bl)} = 1-\rho^{\Theta(bl+|R|)}$ (where the constant behind the Θ-notation is allowed to depend on ϕ, d, and θ). In what follows, we need only show that at least $\frac{\phi}{4}|R|$ registers in $N(R)$ contain a 0 whenever inequality 2.12 holds.

Consider any good dense block B_{i1} in $\mathcal{N}_i$. Since B_{i1} contains more 0-outputs than could possibly come from its d neighboring blocks through the d θl-approximate-sorting circuits, B_{i1} must have contained 0s throughout all the levels of $\mathcal{N}_i$. Hence, by observation 2, each neighboring block of B_{i1} must contain at least $(1-\theta)l$ 0s right after being compared with B_{i1}. Moreover, by observation 1, each of these neighboring blocks of B_{i1} contains at least $(1-\theta d)l$ 0s at the end of $\mathcal{N}$ since they may lose at most $(d-1)\theta l$ 0s through the later $d-1$ or fewer θl-approximate-sorting circuits.

Now, assume for the purposes of contradiction that $b'_{i1} \geq \frac{n_i}{2l(\phi+1)}$ for some i. Then, we can choose $\frac{n_i}{2l(\phi+1)}$ good dense blocks in $\mathcal{N}_i$. By the expansion property of the ϕ-divider, these blocks have at least $\frac{\phi n_i}{2l(\phi+1)}$ neighboring blocks in the top half of $\mathcal{N}_i$. By the discussion of the preceding paragraph, each of these neighboring blocks has at least $(1-\theta d)l$ 0-outputs at the end of $\mathcal{N}_i$. Thus, the number of 0-outputs of $\mathcal{N}_i$ is at least

$$\frac{\phi n_i}{2l(\phi+1)}(1-\theta d)l = \frac{\phi-1}{2(\phi+1)}n_i, \tag{2.13}$$

where we have used equality 2.11. When ϕ is large enough, the quantity in equation 2.13 is strictly larger than $\frac{49}{100}n_i$, which is larger than the number of 0-inputs to $\mathcal{N}_i$. This is a contradiction. Hence, we conclude that for all i

$$b'_{i1} < \frac{n_i}{2l(\phi+1)}. \tag{2.14}$$

By inequality 2.14 and the fact that each $\mathcal{N}_i$ is constructed from a $(\frac{1}{\phi+1}, \phi)$-expander, all of the $\sum_{1\leq i\leq s} b'_{i1}$ good dense blocks have at least $\phi\sum_{1\leq i\leq s} b'_{i1}$ neighboring blocks of top registers in $N(R)$. By the argument used for deriving inequality 2.13, we know that the number of 0s contained in these $\phi\sum_{1\leq i\leq s} b'_{i1}$ top blocks of registers in $N(R)$ is at least

$$\begin{aligned}
& \phi\textstyle\sum_{1\leq i\leq s} b'_{i1}(1-\theta d)l \\
= \;& (\phi-1)\textstyle\sum_{1\leq i\leq s} b'_{i1}l && \text{(by equality 2.11)} \\
\geq \;& \tfrac{2}{3}(\phi-1)\textstyle\sum_{1\leq i\leq s} b_{i1}l && \text{(by inequality 2.12)}
\end{aligned}$$

$$
\begin{aligned}
&\geq \tfrac{1}{3}(\phi-1)\textstyle\sum_{1\leq i\leq s}(b_{i1}l+\frac{b_{i2}l}{\phi}) && \text{(by the assumption of Case 1)}\\
&= \tfrac{1}{3}(\phi-1)\textstyle\sum_{1\leq i\leq s}(b_{i1}l+b_{i2}d\theta l) && \text{(by equality 2.11)}\\
&\geq \tfrac{\phi-1}{3}|R| && \text{(by the definitions of } b_{i1} \text{ and } b_{i2})\\
&\geq \tfrac{\phi}{4}|R| && \text{(for } \phi \text{ sufficiently large).}
\end{aligned}
$$

Case 2: $\sum_{1\leq i\leq s} b_{i1} < \frac{1}{\phi}\sum_{1\leq i\leq s} b_{i2}$.

By Lemma 2.3.1 and a standard Chernoff bound argument, we know that the following inequality holds with probability at least $1-\rho^{\Theta(\sum_{1\leq i\leq s} b_{i2}l)} = 1-\rho^{\Theta(bl+|R|)}$ provided that ρ is sufficiently small:

$$\sum_{1\leq i\leq s} b'_{i2} \geq \frac{2}{3}\sum_{1\leq i\leq s} b_{i2}. \tag{2.15}$$

Next, we show that at least $\frac{\phi}{4}|R|$ registers in $N(R)$ contain a 0 provided that inequality 2.15 holds.

By observation 2, for each good sparse block, its unique neighboring block at the end of $\mathcal{N}$ contains at least $(1-\theta)l$ 0s. Moreover, since the dividers are constructed from d-regular bipartite expanders, different blocks have different neighboring blocks at the end of $\mathcal{N}$. Hence, the number of 0-outputs contained in $N(R)$ is at least

$$
\begin{aligned}
&\textstyle\sum_{1\leq i\leq s} b'_{i2}(1-\theta)l\\
&\geq \tfrac{2}{3}\textstyle\sum_{1\leq i\leq s} b_{i2}(1-\theta)l && \text{(by inequality 2.15)}\\
&\geq \tfrac{1-\theta}{3}\textstyle\sum_{1\leq i\leq s}(b_{i2}l+b_{i1}\phi l) && \text{(by the assumption of Case 2)}\\
&= \tfrac{(1-\theta)\phi}{3}\textstyle\sum_{1\leq i\leq s}(b_{i1}l+b_{i2}d\theta l) && \text{(by equality 2.11)}\\
&\geq \tfrac{(1-\theta)\phi}{3}|R| && \text{(by the definitions of } b_{i1} \text{ and } b_{i2})\\
&\geq \tfrac{\phi}{4}|R| && \text{(for } \phi \text{ sufficiently large).}
\end{aligned}
$$

■

Lemma 2.3.3 *Let $b_{i(1)} \leq \cdots \leq b_{i(\frac{p}{4})}$ be a subsequence of a positive non-decreasing* *sequence $b_1 \leq \cdots \leq b_p$. Then, there exists an integer $s \geq \frac{p}{8}$ such that*

$$\sum_{1 \leq j \leq s} b_{i(j)} \geq \frac{1}{8} \sum_{1 \leq j \leq i(s)} b_j.$$

Proof: Take the minimum s such that

$$i(s) \leq 8\,(s - \frac{p}{8}). \tag{2.16}$$

(Such an s exists because $s = \frac{p}{4}$ satisfies inequality 2.16.) By the minimality of s,

$$i(t) > 8\;(t - \frac{p}{8}) \tag{2.17}$$

for all t such that $\frac{p}{8}+1 \leq t \leq s-1$. By the monotonicity of the sequence $b_1 \leq \cdots \leq b_p$ and inequality 2.17,

$$b_{i(t)} \geq \frac{1}{8} \sum_{8(t-\frac{p}{8}-1)<j\leq 8(t-\frac{p}{8})} b_j \tag{2.18}$$

for all t such that $\frac{p}{8}+1 \leq t \leq s-1$. By inequality 2.16, we have $i(s)-8\,(s-\frac{p}{8}-1) \leq 8$. Hence,

$$b_{i(s)} \geq \frac{1}{8} \sum_{8(s-\frac{p}{8}-1)<j\leq i(s)} b_j. \tag{2.19}$$

Adding inequality 2.19 with inequalities 2.18 (i.e., for all t), we obtain

$$\sum_{\frac{p}{8}<j\leq s} b_{i(j)} \geq \frac{1}{8} \sum_{1 \leq j \leq i(s)} b_j,$$

which is actually stronger than the claimed inequality. ■

Proof of Theorem 2.3.1: We focus on a particular faulty partial l-AKS circuit that violates inequality 2.4 on a particular input permutation Π, and we prove that the faulty circuit has certain properties that can be satisfied by a randomly generated faulty partial l-AKS circuit with probability at most $\rho^{\Theta(l \log m)}$.

We choose the first stage t during which inequality 2.4 is not satisfied at a certain node X. By the minimality of t, we have

$$P_\sigma(Y) \leq \mu \,\mathrm{cap}(Y) \tag{2.20}$$

for any l-AKS tree node Y before stage t. Let Y_i denote the number of i-strange items in node Y. Then, the potential function at node Y can be written as

$$P_\sigma(Y) = \sum_{k \geq 1} \sum_{r \in Y,\, s(r)=k} \sigma^{k-1} = \sum_{k \geq 1} Y_k \sigma^{k-1}. \tag{2.21}$$

Therefore, inequality 2.20 can be rewritten as

$$\sum_{i \geq 1} Y_i \sigma^{i-1} \leq \mu \,\mathrm{cap}(Y).$$

Thus,

$$\sum_{i \geq j} Y_i \sigma^{i-1} \leq \mu \,\mathrm{cap}(Y)$$

for all $j \geq 1$. Since $\sigma > 1$, the previous inequality implies that

$$\frac{\sum_{i \geq j} Y_i}{\mathrm{cap}(Y)} \leq \left(\frac{1}{\sigma}\right)^{j-1} \mu \tag{2.22}$$

for all $j \geq 1$. Inequality 2.22 gives an upper bound on the ratio of the number of items at node Y with strangeness j or more to the capacity of Y, and it will be useful when we upper bound the number of strange items inductively. (It is analogous to inequality 2 in [20].)

On the other hand, by the assumption that $P_\sigma(X) > \mu \,\mathrm{cap}(X)$ and equation 2.21,

$$\sum_{k \geq 1} X_k \sigma^{k-1} > \mu \,\mathrm{cap}(X),$$

where X_k denotes the number of k-strange items in X. Therefore, there exists an integer $k \geq 1$ such that

$$X_k > \left(\frac{1}{2}\right)^k \left(\frac{1}{\sigma}\right)^{k-1} \mu \,\mathrm{cap}(X). \tag{2.23}$$

We choose the minimum k that satisfies inequality 2.23 and analyze how these X_k k-strange items are misplaced into X. By doing so, we derive some necessary properties on the faulty l-AKS circuit. Then, we prove that such properties can be satisfied with probability at most $\rho^{\Theta(l \log m)}$. We will consider two cases: $k = 1$ and $k > 1$. The case $k = 1$ is the hard case in [20] and proceeds without much additional work once we have Lemma 2.3.2. Unfortunately, the case $k > 1$ requires much more work than its fault-free counterpart in [20].

Case 1: $k = 1$.

At the beginning of the first stage, all items are either at the root or the cold storage, and nothing is strange. Hence, our choice of t guarantees $t \geq 2$. We trace back how the X_1 1-strange items at node X are moved into X from P, C, and C', the parent and two children of X respectively. It is easy to see that

$$\begin{aligned} X_1 &= |\{i : i \text{ is 1-strange in } X, \text{ and } i \text{ comes from } C \text{ or } C'\}| \\ &\quad + |\{i : i \text{ is 1-strange in } X, \text{ and } i \text{ comes from } P\}|. \end{aligned} \tag{2.24}$$

Since a 1-strange item in X is 2-strange in either C or C', we can upper bound the first term of equation 2.24 by $C_2 + C_2'$, where C_2 and C_2' denote the number of 2-strange items in C and C' respectively. By inequality 2.22, we have

$$C_2 \leq \frac{1}{\sigma}\mu \operatorname{cap}(C)$$

and

$$C_2' \leq \frac{1}{\sigma}\mu \operatorname{cap}(C').$$

Hence, the first term of equation 2.24 is at most

$$\frac{1}{\sigma}\mu \operatorname{cap}(C) + \frac{1}{\sigma}\mu \operatorname{cap}(C') = \frac{2}{\sigma}\mu \operatorname{cap}(X)\frac{A}{\nu}. \tag{2.25}$$

In what follows, we will use Paterson's argument aided by Lemma 2.3.2 to upper bound the second term of equation 2.24. This is fairly complicated because we have to deal with items that become strange for the first time (in other words, some items may not be strange in P but may be strange in X). An item can be misplaced into X due to one of the following two reasons: (1) the first round divider at P may not divide all the items into the correct halves; (2) P may contain too many items that want to go to X's sibling, X'.

We assume that the number of items that are misplaced into X due to the first reason (i.e., those that are output into the wrong half by the first round divider at P) is equal to

$$\zeta|P| \leq \zeta\frac{1}{2A\nu}\operatorname{cap}(X). \tag{2.26}$$

Here, we use $|P|$, the number of registers in P, instead of cap(P) since P may not be full and we will apply Lemma 2.3.2 where $|P|$, instead of cap(P), will be a parameter.

We next upper bound the number of items that are misplaced into X due to the second reason (i.e., those that want to go to X' from P but will be forced into X due to capacity constraint). Let V be the set of all items of strangeness 0 with respect to X' (some of these items may not be located in node X). Following the notions of Paterson, the "natural" positions for V correspond to the subtree rooted at X' plus one half of P, one-eighth of P's grandparent, and so on. (We can assume an infinite chain of ancestors for this argument. Also, note that the levels of the l-AKS tree are alternatively empty and full and that X' is empty at stage $t-1$.) Ideally, if all items in V are in V's "natural" positions, then P cannot contain too many items that want to go to X'. In reality, some of the items in V may not be in G's "natural" positions. In such a case, some of G's "natural" positions must be occupied by items not in V. In particular, it is not hard to see that the number of items that belong to V but will be forced into X due to capacity constraint is equal to the number of items that are not in V but are occupying V's "natural" positions not in P. We next upper bound the former quantity by giving an upper bound for the latter quantity.

Clearly, an item not in V is 2 or more strange in a child of X', 4 or more strange in a greatgrandchild of X', and so on. Since inequality 2.22 holds before we find the first violation of inequality 2.4 at stage t, a child of X' can contain at most

$$\mu\left(\frac{1}{\sigma}\right)\frac{A}{\nu}\,\mathrm{cap}(X)$$

items not in V, a greatgrandchild of X' can contain at most

$$\mu\left(\frac{1}{\sigma}\right)^3\frac{A^3}{\nu}\,\mathrm{cap}(X)$$

items not in V, and so on. Hence, the total number of items not in V but occupying V's "natural" positions (strictly) below X' is thus upper bounded by

$$\begin{aligned}&2\mu\left(\frac{1}{\sigma}\right)\frac{A}{\nu}\,\mathrm{cap}(X)+8\mu\left(\frac{1}{\sigma}\right)^3\frac{A^3}{\nu}\,\mathrm{cap}(X)+32\mu\left(\frac{1}{\sigma}\right)^5\frac{A^5}{\nu}\,\mathrm{cap}(X)+\cdots\\ \le\;&\frac{2\mu A\,\mathrm{cap}(X)}{\sigma\nu(1-4(\frac{A}{\sigma})^2)}.\end{aligned}\tag{2.27}$$

On the other hand, V's "natural" positions strictly above P may be fully occupied by items not in V, but the number of such positions is at most

$$\frac{\mathrm{cap}(X)}{2^3\nu A^3}+\frac{\mathrm{cap}(X)}{2^5\nu A^5}+\frac{\mathrm{cap}(X)}{2^7\nu A^7}+\cdots=\frac{\mathrm{cap}(X)}{\nu(8A^3-2A)}.\tag{2.28}$$

By the argument of the preceding paragraph, the number of items in P that want to go to X' but will be forced into X due to capacity constraint is upper bounded by the the sum of the quantities in equations 2.27 and 2.28:

$$\frac{2\mu A\,\mathrm{cap}(X)}{\nu\sigma(1-4\frac{A^2}{\sigma^2})}+\frac{\mathrm{cap}(X)}{\nu(8A^3-2A)}.\tag{2.29}$$

Now, adding the quantities in equations 2.25, 2.26, and 2.29, we obtain an upper bound for X_1 in equation 2.24:

$$X_1\le\frac{2\mu A\,\mathrm{cap}(X)}{\sigma\nu}+\frac{2\mu A\,\mathrm{cap}(X)}{\nu\sigma(1-4\frac{A^2}{\sigma^2})}+\frac{\mathrm{cap}(X)}{\nu(8A^3-2A)}+\frac{\zeta\,\mathrm{cap}(X)}{2A\nu}.$$

(We remark that a corresponding formula in [20] contains a term of $\frac{\mu\,\mathrm{cap}(X)}{\nu A}$, which does not appear in our formula. Such a difference occurs since we count the number of items with strangeness exactly 1 whereas Paterson counted the number of items with strangeness at least 1.) By inequality 2.23,

$$X_1 > \frac{\mu \operatorname{cap}(X)}{2}.$$

Combining the last two inequalities, we obtain

$$\frac{2\mu A}{\sigma\nu} + \frac{2\mu A}{\nu\sigma(1-4\frac{A^2}{\sigma^2})} + \frac{1}{\nu(8A^3-2A)} + \frac{\zeta}{2\nu A} > \frac{\mu}{2}. \tag{2.30}$$

Hence,

$$\zeta > \nu\mu A - \frac{4\mu A^2}{\sigma}\left(1 + \frac{1}{1-\frac{4A^2}{\sigma^2}}\right) - \frac{1}{4A^2-1}. \tag{2.31}$$

By choosing ϕ sufficiently large, we can ensure that

$$\frac{4\mu A^2}{\sigma}\left(1 + \frac{1}{1-\frac{4A^2}{\sigma^2}}\right) < \frac{1}{576}.$$

By equations 2.1 and 2.2 and inequality 2.31,

$$\zeta > \frac{43}{576} - \frac{1}{576} - \frac{1}{35} > \frac{4}{100}.$$

By the choice of ζ in inequality 2.26, the preceding inequality implies that the partitioner at P outputs at least $\frac{4}{100}|P|$ of the items into the wrong half. Hence, at the end of the first round divider at P, at least $\frac{4}{100}|P|$ items are output into the wrong half. Without loss of generality, we assume that at the end of the first round divider, at least $\frac{4}{100}|P|$ items that belong to the top half are output to the bottom half. Among the $\frac{4}{100}|P|$ or more items that belong to the top half but are output to the bottom half, at most $\frac{1}{100}|P|$ have ranks greater than $\frac{49}{100}|P|$, and all of the other at least $\frac{4}{100}|P| - \frac{1}{100}|P| \geq \frac{|P|}{100}$ items have ranks $\frac{49}{100}|P|$ or less. Let us define a pair (P, R) to be *bad* if R is a set of bottom registers at the end of the first round divider at P such that $|R| = \frac{|P|}{100}$ and R contains output items with ranks $\frac{49}{100}|P|$ or less at the end of the first round divider at P. The above arguments show that the faulty partial l-AKS circuit violating inequality 2.4 on input permutation Π has a bad pair (P, R). So to complete our analysis of Case 1, we need only show that the probability there exists a bad pair is at most $\rho^{\Theta(l \log m)}$.

For any fixed pair (P, R) such that $|R| = \frac{|P|}{100}$, by Lemma 2.3.2 with $s = 1$, $k = \frac{49}{100}|P|$, and ϕ sufficiently large so that $|R| + \frac{\phi}{4}|R| > k$,

$$\text{Prob}((P, R) \text{ is bad}) \leq \rho^{\Theta(|R|)} = \rho^{\Theta(|P|)}. \qquad (2.32)$$

Hence,

$$\begin{aligned}
&\text{Prob}(\exists \text{ a bad pair } (P, R)) \\
\leq\ & \textstyle\sum_P \sum_{\substack{R \subset P \\ |R| = \frac{|P|}{100}}} \text{Prob}((P, R) \text{ is bad}) \\
\leq\ & \textstyle\sum_P \binom{|P|}{\frac{|P|}{100}} \rho^{\Theta(|P|)} && \text{(by inequality 2.32)} \\
\leq\ & \textstyle\sum_P 2^{|P|} \rho^{\Theta(|P|)} && \text{(since } \tbinom{x}{y} \leq 2^x\text{)} \\
\leq\ & \textstyle\sum_P \rho^{\Theta(|P|)} && \text{(for } \rho \text{ sufficiently small)} \\
\leq\ & O(m \log m) \rho^{\Theta(|P|)}, && (2.33)
\end{aligned}$$

where the last inequality holds because there are at most $O(m \log m)$ l-AKS tree nodes. On the other hand, since priority is given to the upward movement of registers in any l-AKS tree node and since X is not empty at stage t, P contains at least $\lambda\,\text{cap}(P)$ registers at stage $t-1$. Therefore, $|P| \geq \lambda\,\text{cap}(P) \geq \lambda\sqrt{m}$ (since any node in the l-AKS tree for the partial l-AKS circuit has capacity at least $\sqrt{m}$). Hence,

$$O(m \log m)\ \rho^{\Theta(|P|)} \leq O(m \log m) \rho^{\Theta(\lambda\sqrt{m})} \leq \rho^{\Theta(l \log m)}, \qquad (2.34)$$

where the last inequality holds due to the assumption that $l \leq m^{\frac{1}{8}}$ and the fact that λ is a constant. Combining inequalities 2.33 and 2.34 completes the proof for Case 1.

Case 2: $k > 1$.

This case is much more complicated than the preceding one. The source of the difficulty is that all previously known analyses of the AKS circuit rely on the expansion property for arbitrarily small sets of registers and, as discussed in Section 2.1, such an expansion property cannot be preserved with high probability in the presence of faults. To get around this problem, we will trace back $k-1$ stages to see how the X_k k-strange items in X are eventually moved into X. Strange items can come either from below or above in the tree. In the former case, the items would be more strange one stage earlier, and we can apply inequality 2.22. In the latter case, if a good number of the comparators associated with the items work correctly, we will get a certain expansion property. Our hope is to show that even under the loss of the local expansion property for possibly many small sets of registers, globally, the probability that the circuit can lose the expansion property in very many places is relatively small.

Since we have k-strange items in the l-AKS tree at the beginning of stage t, we have $t \geq k+1$ (otherwise all the nodes with depth more than k in the l-AKS tree would be empty at stage t and no item could be k-strange). In what follows, we will trace backward $k-1$ stages and see how these X_k k-strange items are misplaced into X. We will inductively define a sequence of sets $R_t, \ldots, R_{t-k+1}$ such that each R_{t-i} is a set of registers at the beginning of stage $t-i$ with strangeness at least $k-i$. For ease of notation, let $c(R)$ be the set of all the items contained in R for any set of registers R, and let

$$r_{t-i} = |R_{t-i}|. \tag{2.35}$$

Base step: Take R_t as the set of all X_k registers at the beginning of stage t that are k-strange in X. By definition, all the registers in R_t have strangeness at least k.

Inductive step: Assuming that R_{t-i} has been defined for some $0 \leq i < k-1$, we now define R_{t-i-1}. For each register r in an l-AKS tree node X, $c(r)$ may come from either the parent of X or a child of X. In the former case, we say that $c(r)$ *comes from above*; in the latter case, we say that $c(r)$ *comes from below*. Let

$$\alpha_{t-i} = \frac{|\{\text{items in } c(R_{t-i}) \text{ that come from above}\}|}{r_{t-i}},$$

where r_{t-i} is defined in equation 2.35. Given R_{t-i}, a set of registers with strangeness at least $k-i$ at the beginning of stage $t-i$, we construct a set of registers R_{t-i-1} at the beginning of stage $t-i-1$ as follows.

Case (a): $\alpha_{t-i} \le \frac{1}{2}$ (at most half of the items in R_{t-i} come from above). We simply choose R_{t-i-1} as the set of all the r_{t-i} registers at the beginning of stage $t-i-1$ that contain an item in $c(R_{t-i})$.

Case (b): $\alpha_{t-i} > \frac{1}{2}$ (more than half of the items in R_{t-i} come from above). We focus on the $\alpha_{t-i}r_{t-i}$ items in $c(R_{t-i})$ that come from above. By the induction hypothesis, each of these $\alpha_{t-i}r_{t-i}$ items is at least $(k-i)$-strange in a register of R_{t-i}. Hence, each is either too small or too large for the node that it currently resides in. Without loss of generality, we assume that more than half of the items are too small for the nodes that they currently reside in. Let *rank*(i) be the maximum of these small items, and let $c(r)$ be any one of these small items. Let W be the AKS-tree node that contains $c(r)$ at stage $t-i-1$. By the choice of $c(r)$, $c(r)$ is moved from W to a son of W from stage $t-i-1$ to stage $t-i$. Hence, at the end of stage $t-i-1$, $c(r)$ cannot possibly be located in the uppermost or lowermost $\frac{\lambda}{2}$-fraction of the partitioner at W. This means that, at the end of stage $t-i-1$, $c(r)$ is at one of the following four regions of the partitioner at W: (1) the bottom half; (2) the second $\frac{1}{4}$; (3) the second $\frac{1}{8}$; (4) the second $\frac{1}{16}(=\frac{\lambda}{2})$.

If W is a leaf of the l-AKS tree at stage $t-i-1$, it is possible that W is only partially full. In such a case, since priority is given to upward movement of registers in W, at the end of stage $t-i-1$, $c(r)$ cannot possibly be located in the left most $\frac{\lambda}{2}\text{cap}(W)$ (as opposed to $\frac{\lambda}{2}|W|$ where $|W|$ denotes the number of registers in W) registers in W.

We choose the smallest $h \le 4$ such that at least $\frac{1}{8}\alpha_{t-i}r_{t-i}$ of these items are from the hth region (recall that we have assumed that at least $\frac{1}{2}\alpha_{t-i}r_{t-i}$ of the items from above are too small). We trace back where these $\frac{1}{8}\alpha_{t-i}r_{t-i}$ or more items are located at the end of the hth round dividers of the corresponding partitioners. Let U_{t-i} be the set of registers containing these items at the end of the hth round dividers. Let u'_{t-i} be the number of registers in $N(U_{t-i})$ that contain an item less than or equal to *rank*(i) (the meaning of $N(R)$, where R denotes a set of registers in a set of dividers, can be found immediately before Lemma 2.3.2). Note that such registers are at least $(k-i-1)$-strange. If $u'_{t-i} \geq r_{t-i}$, let R_{t-i-1} be any set of registers at the beginning of stage $t-i-1$ that contain $\min\{\phi, \lfloor \frac{u'_{t-i}}{r_{t-i}} \rfloor\} r_{t-i}$ of the u'_{t-i} items. If $u'_{t-i} < r_{t-i}$, then we abandon everything that has been established in Case (b) and simply use the method of Case (a) to choose R_{t-i-1}. By the discussion of the preceding paragraph, we can easily check the correctness of the following claim.

Claim 2.3.1 *Suppose that $\mathcal{D}$ is the hth round divider at an AKS tree node W and $\mathcal{D}$ contains at least one register in U_{t-i}. Then $\mathcal{D}$ is a divider with strictly more than* $\frac{\lambda}{2}\text{cap(W)}$ *registers.*

By the induction hypothesis, the items in $c(R_{t-i})$ are at least $(k-i)$-strange in the registers in R_{t-i}. Since each register in R_{t-i-1} is at most one level higher in the l-AKS tree than its corresponding register in R_{t-i}, all of the registers in R_{t-i-1} are at least $(k-i-1)$-strange. This finishes our inductive construction of R_{t-i-1} from R_{t-i}.

For $0 \leq i \leq k-2$, let

$$\phi_{t-i} = \frac{r_{t-i-1}}{r_{t-i}}. \tag{2.36}$$

Claim 2.3.2 *For $0 \leq i \leq k-2$, ϕ_{t-i} is an integer in $[1, \phi]$.*

Proof: Straightforward from equation 2.36 and the construction of R_{t-i-1} from R_{t-i}. ■

For $0 \leq i \leq k-2$, the strangeness of any item in R_{t-i-1} is at least its strangeness in R_{t-i} minus 1. So by counting all of the elements from above or below, we have

$$P_\sigma(R_{t-i-1}) \geq \phi_{t-i}\sigma^{-1}P_\sigma(R_{t-i})$$

for all i, $0 \leq i \leq k-2$. If $\alpha_{t-i} \leq \frac{1}{2}$, then, by the fact that the elements from below should be one more strange in R_{t-i-1} than in R_{t-i},

$$P_\sigma(R_{t-i-1}) \geq \frac{\sigma}{2}P_\sigma(R_{t-i})$$

for all i such that $0 \le i \le k-2$ and $\alpha_{t-i} \le \frac{1}{2}$. From the preceding pair of inequalities, we have

$$\begin{aligned} P_\sigma(R_{t-k+1}) &\ge P_\sigma(R_t) \prod_{0\le i\le k-2,\ \alpha_{t-i}\le\frac{1}{2}} \left(\frac{\sigma}{2}\right) \prod_{0\le i\le k-2,\ \alpha_{t-i}>\frac{1}{2}} \left(\frac{\phi_{t-i}}{\sigma}\right) \\ &\ge \mu\,\mathrm{cap}(X) \prod_{0\le i\le k-2,\ \alpha_{t-i}\le\frac{1}{2}} \left(\frac{\sigma}{2}\right) \prod_{0\le i\le k-2,\ \alpha_{t-i}>\frac{1}{2}} \left(\frac{\phi_{t-i}}{\sigma}\right). \end{aligned} \tag{2.37}$$

Since we start from a set of registers in node X and we move at most one level upward and one level downward in the l-AKS tree when constructing R_{t-i-1} from R_{t-i}, R_{t-k+1} is located within $k-1$ levels from X. Therefore, the total capacity of all nodes that can possibly contain registers in R_{t-k+1} is upper bounded by

$$\begin{aligned} &\mathrm{cap}(X)\ \nu^{-(k-1)} \left[(2A)^{k-1} + (2A)^{k-2} + \ldots + A + 1 + A^{-1} \ldots + A^{-(k-2)} + A^{-(k-1)}\right] \\ < &\ \mathrm{cap}(X) \left(\frac{2A}{\nu}\right)^{k-1} \frac{1}{1-\frac{1}{2A}}. \end{aligned} \tag{2.38}$$

By inequality 2.38 and the fact that inequality 2.4 is not yet violated at stage $t-k+1$,

$$P_\sigma(R_{t-k+1}) \le \mu\ \mathrm{cap}(X) \left(\frac{2A}{\nu}\right)^{k-1} \frac{1}{1-\frac{1}{2A}}. \tag{2.39}$$

Combining inequalities 2.37 and 2.39, we get

$$\prod_{0\le i\le k-2,\, \alpha_{t-i}\le\frac{1}{2}} \left(\frac{\sigma}{2}\right) \prod_{0\le i\le k-2,\, \alpha_{t-i}>\frac{1}{2}} \left(\frac{\phi_{t-i}}{\sigma}\right) \le \left(\frac{2A}{\nu}\right)^{k-1} \frac{1}{1-\frac{1}{2A}}. \tag{2.40}$$

Hence,

$$\prod_{0\le i\le k-2,\, \alpha_{t-i}>\frac{1}{2}} \phi_{t-i} \le \left(\frac{4A}{\nu}\right)^{k-1} \frac{1}{1-\frac{1}{2A}} \sigma^{x-y}, \tag{2.41}$$

where $x = |\{i : 0 \le i \le k-2,\ \alpha_{t-i} > \frac{1}{2}\}|$ and $y = |\{i : 0 \le i \le k-2,\ \alpha_{t-i} \le \frac{1}{2}\}|$. From equality 2.3, the fact $\phi_{t-i} \ge 1$ (see Claim 2.3.2), and inequality 2.41, we obtain

$$\left(\frac{4\sigma A}{\nu(1-\frac{1}{2A})}\right)^{2z} = \left(\frac{\phi}{65}\right)^{z} \le \prod_{0\le i\le k-2,\, \alpha_{t-i}>\frac{1}{2}} \phi_{t-i} \le \left(\frac{4A}{\nu}\right)^{k-1} \frac{1}{1-\frac{1}{2A}} \sigma^{x-y}, \tag{2.42}$$

where $z = |\{i : 0 \le i \le k-2,\ \alpha_{t-i} > \frac{1}{2},\ \phi_{t-i} \ge \frac{\phi}{65}\}|$. Let β be a constant such that

$$\frac{4A}{\nu(1-\frac{1}{2A})} = \sigma^{\beta}. \tag{2.43}$$

From inequality 2.42 and the fact that $k \ge 2$,

$$\sigma^{(\beta+1)2z} \le \sigma^{(x-y)+\beta(k-1)}.$$

Therefore,

$$z \le f(\beta) = \frac{\beta(k-1)+(x-y)}{2(1+\beta)}. \tag{2.44}$$

By simple calculus, we have

$$f'(\beta) = \frac{(k-1)-(x-y)}{2(1+\beta)^2} \ge 0.$$

Thus, $f(\beta)$ is monotonically increasing. According to equation 2.43, we can enforce $\beta \le \frac{1}{2}$ by letting σ be large. Therefore, by inequality 2.44,

$$z \le f\left(\frac{1}{2}\right) = \frac{\frac{1}{2}(k-1)+x-y}{3} \le \frac{x-y}{2} + \frac{k-1}{4},$$

for σ sufficiently large. Thus,

$$\begin{aligned} & |\{i : 0 \le i \le k-2,\ \alpha_{t-i} > \frac{1}{2},\ \phi_{t-i} < \frac{\phi}{65}\}| \\ = \ & x - z \ge x - \left(\frac{x-y}{2} + \frac{k-1}{4}\right) = \frac{k-1}{4}, \end{aligned} \tag{2.45}$$

where the last equality holds since $x + y = k - 1$ by definition. Recall that for any i such that $0 \le i \le k-2$ and $\alpha_{t-i} > \frac{1}{2}$, we have used U_{t-i} as an intermediate group of registers in constructing R_{t-i-1} from R_{t-i} (see page 41). We define U_{t-i} to be *bad* if $\alpha_{t-i} > \frac{1}{2}$ and $\phi_{t-i} < \frac{\phi}{65}$. Inequality 2.45 implies that the number of bad U_{t-i} is at least $\frac{k-1}{4}$.

Let $i(1) < \cdots < i(q)$ be the increasing sequence of all integers i such that $0 \le i \le$

$k-2$ and U_{t-i} is bad. We have $q \geq \frac{1}{4}(k-1)$. Applying Lemma 2.3.3 to the sequences

$$r_t \leq r_{t-1} \leq \cdots \leq r_{t-(k-2)} \text{ and } r_{t-i(1)} \leq r_{t-i(2)} \leq \cdots \leq r_{t-i(q)},$$

we know that there exists an integer $s \geq \frac{k-1}{8}$ such that

$$\sum_{1 \leq j \leq s} r_{t-i(j)} \geq \frac{1}{8} \sum_{0 \leq j \leq i(s)} r_{t-j}. \tag{2.46}$$

We will finish our analysis of Case 2 by proving that the probability there exists such a sequence of bad sets (which will be referred to as a *bad sequence* hereafter) $U_{t-i(1)}, \ldots, U_{t-i(s)}$ is small. For $1 \leq j \leq s$, let B_j be the set of blocks that contain at least one register in $U_{t-i(j)}$, $u_j = |U_{t-i(j)}|$, and $b_j = |B_j|$, i.e., the number of blocks that contain at least one register in $U_{t-i(j)}$.

Claim 2.3.3 *For a given sequence* $U_{t-i(1)}, U_{t-i(2)}, \ldots, U_{t-i(s)}$,

$$\text{Prob}\left(U_{t-i(1)}, U_{t-i(2)}, \ldots, U_{t-i(s)} \text{ is a bad sequence}\right) \leq \rho^{\Theta(\sum_{1 \leq j \leq s} l b_j)}.$$

Proof: If $U_{t-i(j)}$ is bad, then $\phi_{t-i(j)} < \frac{\phi}{65}$ by definition. Hence

$$\left\lfloor \frac{u'_{t-i(j)}}{r_{t-i(j)}} \right\rfloor < \frac{\phi}{65}, \tag{2.47}$$

where $u'_{t-i(j)}$ is defined in the construction of $R_{t-i(j)-1}$ from $R_{t-i(j)}$ (see page 41). For a sufficiently large choice of the constant ϕ (so that $\frac{\phi}{65} + 1 < \frac{\phi}{64}$), inequality 2.47 implies

$$\frac{u'_{t-i(j)}}{r_{t-i(j)}} < \frac{\phi}{64}. \tag{2.48}$$

According to our inductive construction of $R_{t-i(j)-1}$ from $R_{t-i(j)}$ and the fact $\alpha_{t-i(j)} > \frac{1}{2}$ (since $U_{t-i(j)}$ is assumed to be bad), we have $|U_{t-i(j)}| \geq \frac{1}{8}\alpha_{t-i(j)} r_{t-i(j)} \geq \frac{r_{t-i(j)}}{16}$. Therefore, inequality 2.48 implies

$$\frac{u'_{t-i(j)}}{|U_{t-i(j)}|} < \frac{\phi}{4}. \tag{2.49}$$

Note that inequality 2.22 holds at stage $t - i(j)$ and that all of the items in $U_{t-i(j)}$ are at least 1-strange. Hence, by Claim 2.3.1 and the fact $\frac{\mu}{\lambda/2} < 0.49$, $U_{t-i(j)}$ satisfies the condition on R in Lemma 2.3.2. Thus, by applying Lemma 2.3.2 to all of the dividers associated with $U_{t-i(j)}$, the probability that inequality 2.49 holds is at most $\rho^{\Theta(l\, b_j)}$. Hence, we have

$$\text{Prob}(U_{t-i(j)} \text{ is bad }) \leq \rho^{\Theta(l\, b_j)}.$$

The claim now follows from the independence of the $U_{t-i(j)}$'s, $1 \leq j \leq s$. ■

In the next few claims, we upper bound the number of possible ways of choosing the sequence $U_{t-i(1)}, \ldots, U_{t-i(s)}$ and show that even after this number is taken into account, the probability that a faulty l-AKS circuit contains a bad sequence $U_{t-i(1)}, \ldots, U_{t-i(s)}$ is very small.

Claim 2.3.4 *For fixed k, the number of ways for choosing the sequence $r_t, r_{t-1}, \ldots,$ r_{t-k+1} is at most $m\phi^{k-1}$.*

Proof: The number of ways for choosing r_t is at most m. By Claim 2.3.2, when r_{t-i} is given, the number of ways for choosing r_{t-i-1} is at most ϕ. Overall, the number of ways for choosing $r_t, r_{t-1}, \ldots, r_{t-k+1}$ is at most $m\phi^{k-1}$. ■

Claim 2.3.5 *When* $\text{cap}(X)$ *and r_t are both given, the number of ways for choosing R_t is at most*

$$O(\sqrt{m}\log m)\binom{\text{cap}(X)}{r_t}.$$

Proof: Since the partial l-AKS circuit is run for $O(\log m)$ stages and the l-AKS tree has at most $(m/\text{cap(root)}) \leq \sqrt{m}$ nonempty nodes at each stage, the total number of ways for choosing node X is upper bounded by

$$O(\sqrt{m}\log m).$$

When X is given, the number of ways for choosing R_t, a set of registers contained in X, is at most

$$\binom{\text{cap}(X)}{r_t}.$$

Multiplying the quantities in the last two formulae, we obtain the desired upper bound. ■

Claim 2.3.6 *When the sequence $r_{t-i(1)}, \ldots, r_{t-i(s)}$ is given, the number of ways for choosing the sequence $b_1, \ldots, b_s$ is at most*

$$O\left(\prod_{1 \le j \le s} r_{t-i(j)}\right).$$

Proof: This is because $b_j \le r_{t-i(j)}$ for each $1 \le j \le s$. ■

Claim 2.3.7 *If R_t and the sequences $r_t, r_{t-1}, \ldots, r_{t-i(s)+1}, r_{t-i(s)}$, $i(1), i(2), \ldots, i(s)$, and $b_1, \ldots, b_s$ are given, then the number of ways for choosing the sequence $B_1, \ldots, B_s$ is at most*

$$2^{O(\sum_{1 \le j \le s} l b_j)}.$$

Proof: Let $\tilde{B}_{t-i}$ be the set of blocks that contain at least one register in R_{t-i} for $i = 0, 1, 2, \ldots, i(s)+1$. We first upper bound the number the possible sequences

$$\tilde{B}_t, \tilde{B}_{t-1}, \ldots, \tilde{B}_{t-i(s)-1}.$$

(Note that by the choice of s immediately before inequality 2.46, $i(s) \le k-2$ and hence $\tilde{B}_{t-i(s)-1}$ is well defined.) Clearly, $\tilde{B}_t$ is fixed when R_t is given. We next count the number of ways for choosing $\tilde{B}_{t-i-1}$ when $\tilde{B}_{t-i}$ is given. Each block in $\tilde{B}_{t-i-1}$ is connected by a $(t-i-1)$th stage partitioner to some block in $\tilde{B}_{t-i}$. Since each block is connected to at most d blocks by a divider, it is connected to at most d^4 blocks by a partitioner. On the other hand, there are at most r_{t-i} blocks in $\tilde{B}_{t-i}$. Therefore, when $\tilde{B}_{t-i}$ is given, the number of ways for choosing $\tilde{B}_{t-i-1}$ is at most

$$\binom{d^4 r_{t-i}}{r_{t-i-1}} = \binom{d^4 r_{t-i}}{\phi_{t-i} r_{t-i}} \le \left(\frac{e\, d^4}{\phi_{t-i}}\right)^{\phi_{t-i} r_{t-i}} \le 2^{O(r_{t-i})}, \qquad \text{for } i = 0, 1, \ldots, i(s) \quad (2.50)$$

where the constant behind the O-notation is dependent on ϕ and d but not dependent on ρ.

On the other hand, B_j corresponds to $U_{t-i(j)}$, and $\tilde{B}_{t-i(j)-1}$ corresponds to $R_{t-i(j)-1}$. Moreover, by the description of U_{t-i} on page 41, $R_{t-i(j)-1}$ and $U_{t-i(j)}$ correspond to the same stage of the AKS tree. Hence, B_j and $\tilde{B}_{t-i(j)-1}$ correspond to registers within the same stage. Therefore, by an argument similar to the preceding paragraph, we know that when $\tilde{B}_{t-i(j)-1}$ is given, the number of ways for choosing B_j is at most

$$2^{O(r_{t-i(j)-1})} = 2^{O(r_{t-i(j)})}, \qquad \text{for } j = 1, 2, \ldots, s, \tag{2.51}$$

where we have used Claim 2.3.2. Multiplying all the quantities (i.e., for $i = 1, 2, \ldots, i(s)$) in inequality 2.50 with all the quantities (i.e., for $j = 1, 2, \ldots, s$) in equation 2.51, we obtain an upper bound on the number of ways to choose the sequence $B_1, B_2, \ldots, B_s$ under the assumption of the claim:

$$\begin{aligned} & 2^{O(\sum_{0\le j\le i(s)} r_{t-j})} \cdot 2^{O(\sum_{1\le j\le s} r_{t-i(j)})} \\ = \;& 2^{O(\sum_{1\le j\le s} r_{t-i(j)})} && \text{(by inequality 2.46)} \\ = \;& 2^{O(\sum_{1\le j\le s} l\, b_j)} && \text{(since } r_{t-i(j)} \le l\, b_j\text{).} \end{aligned}$$

■

Claim 2.3.8 *If the sequence $B_1, \ldots, B_s$ is given, then the probability there exists a bad sequence $U_{t-i(1)}, \ldots, U_{t-i(s)}$ is at most*

$$\rho^{\Theta(\sum_{1\le j\le s} l\, b_j)}. \tag{2.52}$$

Proof: For each j such that $1 \le j \le s$, there are b_j blocks of size l in B_j. Hence, when B_j is given, the number of ways for selecting the elements in $U_{t-i(j)}$ is at most $2^{l\, b_j}$. Thus, the number of ways for selecting the sequence $U_{t-i(1)}, \ldots, U_{t-i(s)}$ is upper bounded by

$$2^{l \sum_{1\le j\le s} b_j}.$$

Therefore,

$$\begin{aligned}
& \text{Prob}(\exists \text{ a bad sequence } U_{t-i(1)}, \ldots, U_{t-i(s)} \mid B_1, \ldots, B_s \text{ are all given}) \\
\leq \;& 2^{\sum_{1\leq j\leq s} l\, b_j} \cdot \text{Prob}(\text{a fixed sequence } U_{t-i(1)}, \ldots, U_{t-i(s)} \text{ is bad}) \\
\leq \;& 2^{\sum_{1\leq j\leq s} l\, b_j} \rho^{\Theta(\sum_{1\leq j\leq s} l\, b_j)} \\
\leq \;& \rho^{\Theta(\sum_{1\leq j\leq s} l\, b_j)},
\end{aligned}$$

where the second inequality follows by Claim 2.3.3, and the last inequality holds for ρ sufficiently small. ■

Claim 2.3.9 *Let* cap(X)*,* k*, and sequences* $r_t, \ldots, r_{t-k+1}$ *and* $i(1), \ldots, i(s)$ *be given. Then*

$$\text{Prob}(\exists \text{ a bad sequence } U_{t-i(1)}, \ldots, U_{t-i(s)}) \leq \rho^{\Theta(l \log m)}.$$

Proof: Claims 2.3.5 to 2.3.8 imply that for given cap(X), k, and sequences $r_t, \ldots, r_{t-k+1}$ and $i(1), \ldots, i(s)$,

$$\begin{aligned}
& \text{Prob}(\exists \text{ a bad sequence } U_{t-i(1)}, \ldots, U_{t-i(s)}) \\
\leq \;& O(\sqrt{m}\, \log m) \binom{\text{cap}(X)}{r_t} O\left(\prod_{1\leq j\leq s} r_{t-i(j)} \right) 2^{O(\sum_{1\leq j\leq s} l\, b_j)} \rho^{\Theta(\sum_{1\leq j\leq s} l\, b_j)} \\
\leq \;& O(\sqrt{m}\, \log m) \binom{\text{cap}(X)}{r_t} \rho^{\Theta(\sum_{1\leq j\leq s} l\, b_j)} \qquad (2.53)
\end{aligned}$$

where the last inequality follows since $l\, b_j \geq r_{t-i(j)}$ and ρ can be chosen sufficiently small.

To prove the claim, we need to show that the quantity in equation 2.53 is at most $\rho^{\Theta(l \log m)}$. Since $s \geq \frac{k-1}{8}$ and $l\, b_j \geq r_{t-i(j)} \geq r_t$ for each j such that $1 \leq j \leq s$, it suffices to show that, for ρ sufficiently small,

$$O(\sqrt{m}\, \log m) \left(\frac{\text{cap}(X)\, e}{r_t} \right)^{r_t} \rho^{\Theta((r_t + l)(k-1))} \leq \rho^{\Theta(l \log m)}.$$

We do this by proving

$$\left(\frac{\mathrm{cap}(X)\,e}{r_t}\rho^{k-1}\right)^{r_t+l} \le \rho^{\Theta(l\log m)}. \tag{2.54}$$

Let C_0 be a constant such that

$$\left(\frac{1}{2}\right)^{C_0\log m+1}\left(\frac{1}{\sigma}\right)^{C_0\log m}\mu\sqrt{m} = m^{\frac{1}{4}}.$$

We establish inequality 2.54 by considering the following two cases.

Case (a): $k-1 < C_0\log m$.

By inequality 2.23 and the fact that $\mathrm{cap}(X) \ge \sqrt{m}$, we have

$$r_t = X_k > \left(\frac{1}{2}\right)^{k}\left(\frac{1}{\sigma}\right)^{k-1}\mu\,\mathrm{cap}(X) \ge \left(\frac{1}{2}\right)^{C_0\log m+1}\left(\frac{1}{\sigma}\right)^{C_0\log m}\mu\sqrt{m} = m^{\frac{1}{4}}.$$

Thus, $r_t+l \ge m^{\frac{1}{4}} \ge l\log m$ since $l \le m^{\frac{1}{8}}$. Therefore, in order to show inequality 2.54, we need only show that for ρ sufficiently small,

$$\frac{\mathrm{cap}(X)\,e}{r_t}\rho^{k-1} < \rho^{\Theta(1)}.$$

For sufficiently small ρ, the last inequality follows since $k \ge 2$ and, by inequality 2.23, $\frac{\mathrm{cap}(X)}{r_t} = \frac{\mathrm{cap}(X)}{X_t} \le \frac{2}{\mu}(2\sigma)^{k-1}$.

Case (b): $k-1 \ge C_0\log m$.

As argued in [20], at any stage in the l-AKS tree, there exists a level L (which might be full or partially full) such that all the levels strictly below L are empty and all the levels strictly above L are alternately full (i.e., either all the nodes at the even levels strictly above L are full or all the nodes at the odd levels strictly above L are full). This immediately implies that each nonempty node has capacity at most $O(m)$. Hence, by the fact that node X is not empty at stage t, we know $\mathrm{cap}(X) = O(m)$. Therefore, for ρ sufficiently small,

$$\frac{\mathrm{cap}(X)\ e}{r_t}\rho^{k-1} \le O(m)e\rho^{C_0\log m} \le \rho^{\Theta(\log m)},$$

establishing inequality 2.54. ■

Continuing with the proof of Theorem 2.3.1, the number of choices for k is at most

$$O(\log m),$$

and (as argued in Case (b) of Claim 2.3.9) the number of ways for choosing cap(X) is at most

$$O(m).$$

By Claim 2.3.4, when k is given, the number of choices for the sequence $r_t, \ldots, r_{t-k+1}$ is at most

$$m\phi^{k-1} \leq O(m)2^{O(\log m)} = 2^{O(\log m)}.$$

Furthermore, when k is given, the number of choices for the sequence $i(1), \ldots, i(s)$ is at most

$$\sum_{\frac{k-1}{8} \leq s \leq k-1} \binom{k-1}{s} \leq 2^{k-1} \leq 2^{O(\log m)},$$

since $\frac{k-1}{8} \leq s \leq k-1$ and $i(1) < i(2) < \cdots < i(s)$. Therefore, the number of choices for k, cap(X), and the sequences $r_t, \ldots, r_{t-k+1}$ and $i(1), \ldots, i(s)$ is at most

$$O(\log m) \cdot O(m) \cdot 2^{O(\log m)} \cdot 2^{O(\log m)} = 2^{O(\log m)}.$$

Multiplying the above number with the quantity in Claim 2.3.9, we obtain the desired upper bound for the probability that a bad sequence $U_{t-i(1)}, \ldots, U_{t-i(s)}$ exists. This completes our consideration of Case 2, as well as the inductive step for proving the theorem. ■

Corollary 2.3.1 *Let $\beta = 1 - \frac{1}{8\log 6}$ and X be the set of all output registers of an m-input partial l-AKS circuit (where $l \leq m^{\frac{1}{8}}$) with ϕ and σ being sufficiently large constants. Under both the passive and the reversal fault models, there exists a fixed partition of X into disjoint sets $\{S, X_1, \ldots, X_{\bar{m}}\}$ where $\bar{m} = \Theta(m^{1-\beta})$, $|S| = O(m^{\frac{3}{4}})$, and $|X_1| = \ldots = |X_{\bar{m}}| = \Theta(m^{\beta})$ such that when ρ is less than a sufficiently small* *constant, a randomly faulty l-AKS circuit has the following property with probability at least* $1-\rho^{\Theta(l\log m)}$*: On all input permutations, the items in* X_i *are smaller than the items in* X_j *for* $1 \le i < j \le \bar{m}$.

Proof: We prove the corollary by exhibiting a fixed partition of X with the necessary properties. Assume that at the last stage of the partial l-AKS circuit, the l-AKS tree has depth d (i.e., nodes at level d are nonempty and nodes strictly below d are all empty). If we were to run the l-AKS circuit for another stage, then the root of the l-AKS tree would have capacity less than $\sqrt{m}$. Hence, the capacity of the root at the last stage of the l-AKS tree is at most $\frac{\sqrt{m}}{\nu}$. Therefore, at the last stage of the partial l-AKS circuit, the total capacity of all the nonempty nodes and the cold storage in the l-AKS tree is at most

$$\begin{aligned} & \frac{\sqrt{m}}{\nu} + \frac{\sqrt{m}}{\nu}\cdot 2A + \ldots + \frac{\sqrt{m}}{\nu}\cdot(2A)^{d-1} + \frac{\sqrt{m}}{\nu 2A} + \frac{\sqrt{m}}{\nu(2A)^2} + \frac{\sqrt{m}}{\nu(2A)^3} + \ldots \\ = & \frac{(2A)^d}{2A-1}\cdot\frac{\sqrt{m}}{\nu} \\ < & 6^d\sqrt{m}, \end{aligned} \tag{2.55}$$

where the last inequality follows from equation 2.1. On the other hand, according to [20], either all the nodes at the even levels strictly above level d are full or all the nodes at the odd levels strictly above level d are full. Hence, by the fact cap(root) $\geq \sqrt{m}$, the number of registers in the l-AKS tree is at least

$$\begin{aligned} & \frac{1}{2A+1}\left(\sqrt{m} + \sqrt{m}\cdot 2A + \ldots + \sqrt{m}\cdot(2A)^{d-2}\right) \\ = & \frac{(2A)^{d-1}-1}{(2A+1)(2A-1)}\sqrt{m} \\ > & 6^{d-3}\sqrt{m}. \end{aligned} \tag{2.56}$$

Combining inequalities 2.55 and 2.56 with the fact that there are actually m registers in the l-AKS tree (including the cold storage), we find that

$$\sqrt{m}\, 6^{d-3} < m < 6^d\sqrt{m},$$

and hence

$$\log_6 \sqrt{m} < d < \log_6 \sqrt{m} + 3. \tag{2.57}$$

Let S be the set of output registers at the top $\lceil \frac{d}{2} \rceil$ levels of the l-AKS tree. Consider the $\lfloor \frac{d}{4} \rfloor$th level of the l-AKS tree (the root is assumed to be at level 1). Label all the $2^{\lfloor \frac{d}{4} \rfloor - 1}$ nodes at this level from left to right with $1, \ldots, \bar{m}$, where

$$\bar{m} = 2^{\lfloor \frac{d}{4} \rfloor - 1} = \Theta(m^{1-\beta}). \tag{2.58}$$

(Here, we have used inequality 2.57.) For $i \le \bar{m}$, let T_i be the set of registers contained in the tree rooted at the node labeled by i. Let

$$X_i = T_i - S.$$

The sets S and X_i, $1 \le i \le \bar{m}$, completely determine the partition $X = S \bigcup X_1 \bigcup \ldots \bigcup X_{\bar{m}}$, and we only need to show that the partition has the claimed property.

By calculations similar to that for equation 2.55 and by inequality 2.57, we have

$$|S| \le 6^{\lceil \frac{d}{2} \rceil} \sqrt{m} = O(m^{\frac{3}{4}}). \tag{2.59}$$

For $i \le \bar{m}$,

$$\begin{aligned} |X_i| &\ge |T_i| - |S| \\ &\ge \tfrac{m-|S|}{\bar{m}} - |S| \\ &\ge \Omega(m^{\beta}) \qquad \text{(by equation 2.58 and inequality 2.59).} \end{aligned}$$

On the other hand, by equation 2.58,

$$|X_i| \le \frac{m}{\bar{m}} = O(m^{\beta}).$$

Thus,

$$|X_i| = \Theta(m^{\beta}),$$

as claimed.

It remains to prove that with probability at least $1 - \rho^{O(l \log m)}$, on all input permutations,

$$x_i < x_j \text{ for any pair of items } x_i \in X_i \text{ and } x_j \in X_j \text{ where } i < j.$$

By Theorem 2.3.1, we need only to show that the above relationship holds provided inequality 2.4 is true. Assume for the purposes of contradiction that for a permutation Π and some $i < j$, there exist items x_i in X_i and x_j in X_j such that $x_i > x_j$. Then, either x_i is not in the natural interval for the root of T_i or x_j is not in the natural interval for the root of T_j. Without loss of generality, we assume that item x_i is not in the natural interval for the root of T_i. Then, x_i is at least $\frac{d}{4}$-strange since x_i is at least $\lceil \frac{d}{2} \rceil - \lfloor \frac{d}{4} \rfloor \geq \frac{d}{4}$ away from the root of T_i. Let r_i be the register that contains x_i. By the definition of the potential function together with inequality 2.57,

$$P_\sigma(r_i) \geq \sigma^{\frac{d}{4}-1} \geq \sigma^{\frac{\log_6 \sqrt{m}}{4}-1}. \tag{2.60}$$

On the other hand, by inequality 2.4,

$$P_\sigma(r_i) \leq P_\sigma(X_i) \leq \mu \operatorname{cap}(X_i) \leq \mu m. \tag{2.61}$$

Inequalities 2.60 and 2.61 now yield a contradiction for a sufficiently large choice of the constant σ. ■

Chapter 3

Constructions of Fault-Tolerant Sorting Circuits, Networks, and EREW Algorithms

In this chapter, we use the fault-tolerance of the l-AKS circuit to construct passive-fault-tolerant sorting circuits, reversal-fault-tolerant sorting networks, and EREW fault-tolerant sorting algorithms. Despite many complicated technical details, we construct all the circuits, networks, and algorithms in the following fashion. We first use a partial l-AKS circuit to move all but $O(n^{\frac{3}{4}})$ special items to within $O(n^{\beta})$ of the correct positions, where $\beta < 1$ is the constant in Corollary 2.3.1. Then, we use an approximate-insertion circuit to be described soon to insert the special items close to their correct positions. After these two steps, all items are within $O(n^{\alpha})$ of the correct position, for some constant $\alpha < 1$, and we can complete the sort recursively. We remark that we will only use 1-AKS circuits except in the constructions for reversal faults, where we do need l-AKS circuits with $l > 1$.

The remainder of the chapter is organized into sections as follows. Section 3.1 contains a passive-fault-tolerant sorting circuit. Section 3.2 contains our results for reversal-fault-tolerant approximate-sorting circuits and reversal-fault-tolerant sorting networks. Section 3.3 contains an optimal EREW PRAM fault-tolerant sorting algorithm. We conclude in Section 3.4 by extending all the results to worst-case faults.

3.1 Passive-Fault-Tolerant Circuits

In this section, we use the fault-tolerance of the 1-AKS circuit to construct circuits that are tolerant to passive faults. The most important result of this section is the construction of a passive-fault-tolerant sorting circuit with $O(\log n \log\log n)$ depth. Since a circuit contains at most $\frac{n}{2}$ comparators at each level, our circuit has $O(n \log n \log\log n)$ size. This provides the first nontrivial upper bound for sorting circuits that tolerate random passive faults, and answers the open question posed by Yao and Yao [26] up to an $O(\log\log n)$ factor.

In [26], Yao and Yao conjectured that any passive-fault-tolerant sorting or merging circuit has $\omega(n \log n)$ size. As an interesting application of our technique for sorting, we prove a tight bound of $\Theta(n \log n)$ on the size of passive-fault-tolerant *selection* circuits. (See page 13 for the definition of a selection circuit and the comments therein.) This result would imply a separation of the complexities for merging and selection if Yao and Yao's conjecture is correct. To the best of our knowledge, no such separation result is currently known for merging and selection.

Theorem 3.1.1 *There exists an explicit construction of a passive-fault-tolerant sorting circuit with $O(\log n \log\log n)$ depth.*

We first prove two lemmas before proving Theorem 3.1.1. An $(m+1)$-input circuit $\mathcal{C}$ is defined to be a Δ-*approximate-insertion* circuit if $\mathcal{C}$ outputs every item to within Δ of the correct position provided that the input sequence consists of a sorted list of m items plus another "unknown" item to be input to a given register.

Lemma 3.1.1 *For any ρ less than a sufficiently small constant, there exists an explicit construction of an $(m+1)$-input $(\rho, \rho^{\Theta(\log m)})$-passive-fault-tolerant $m^{\frac{6}{7}}$-approximate-insertion circuit with $O(\log m)$ depth.*

Proof: As mentioned in the introduction, unless specified otherwise, all the circuits constructed in the thesis are standard. So we will prove the lemma by explicitly constructing a standard circuit with the claimed properties. We will first construct a circuit that receives the unknown item at the top register. A circuit that receives the unknown item at the bottom register can be constructed in an entirely symmetric fashion. We then use these two circuits to construct a circuit that solves the general approximate-insertion problem where the unknown item is input to an arbitrary register in a general position.

In what follows, we describe an $m^{\frac{6}{7}}$-approximate-insertion circuit $\mathcal{C}$ that receives the unknown item at the top register. By the definition of approximate-insertion, the input list consists of a sorted list, L, of length m, and an unknown item, x, which will be input to the top register of $\mathcal{C}$ according to our assumption. Let $t = \lceil \log m \rceil$. Rather than solving the given approximate-insertion problem "directly", we first partition the m items of the sorted list L into $r = \lfloor \frac{m}{s} \rfloor$ contiguous group of size $s = 2^{t - \lfloor t/6 \rfloor}$, except for the last group, which has $m - (r-1)s$ items. Clearly, the last group contains at least s and strictly less than $2s$ items. To construct circuit $\mathcal{C}$, it is useful to think of all of the items in any one group as being indistinguishable. Conceptually, we now solve an insertion problem with only $r+1$ inputs (one for each group, plus the unknown item x), and we only need to move x into the group that x belongs to. This is because each group contains at most $2s$ items and the maximum distance between any pair of items in any single group is at most $2s \le 2 \cdot 2^{\lceil \frac{5}{6} t \rceil} < 4 \cdot 2^{\frac{5}{6} \lceil \log m \rceil} < m^{\frac{6}{7}}$ for sufficiently large m.

We call a register *clean at level i* if it is not the top register and if it has not been compared with any other register strictly before level i; we call a register *unclean at level i* otherwise. Intuitively, a register that is clean at level i has zero chance to contain x at level i; and a register that is unclean at level i has non-zero chance to contain x. We also assign an index to each of the m registers as follows: (1) assign index 0 to the top register; (2) for each j such that $1 \le j \le r-1$, assign index j to all the registers whose inputs are taken from the jth group, which contains items with ranks between $(j-1)s+1$ and js in list L; (3) assign index r to registers whose inputs are taken from the last group, which contains items with ranks between $(r-1)s+1$ and m in list L. Circuit $\mathcal{C}$ will have $5 \log r$ depth and the following properties: for each i such that $1 \le i \le 5\log r$, (1) every comparator at level i has its MIN input to be unclean at level i and its MAX input to be clean at level i, and (2) each register that is unclean at level i is compared with a register with a larger index that is clean at level i. Since $\mathcal{C}$ has $5\log r$ depth, the total number of clean registers required in the whole process is

$$\sum_{1 \le i \le 5\log r} 2^{i-1} = 2^{5\log r} - 1 = r^5 - 1 < \left(\frac{m}{s}\right)^5 \le s,$$

which is smaller than or equal to the number of items contained in any of the groups. Hence, we cannot possibly run out of clean registers in any group.

Given these restrictions, we can complete the description of circuit $\mathcal{C}$ by specifying the unique comparator at the first level and by specifying for each register y that is output from level $i-1$, the index of the register against which y will be compared at level i. In particular, $\mathcal{C}$ is inductively constructed as follows. The unique comparator at the first level connects input x and a register with index 1, which is certainly clean at level 1. The second level of $\mathcal{C}$ will have two comparators. The MIN output of the comparator at the first level is fed into the MIN input of a comparator whose MAX input takes a register with index 1 that is clean at level 2; The MAX output of the comparator at the first level is fed into the MIN input of a comparator whose MAX input takes a register with index $1+2=3$ that is clean at level 2. In general, for each comparator at level $i-1$ that connects two inputs with indices j and $j+h$, the MAX output (which certainly has index $j+h$) is fed into the MIN input of a comparator at level i whose MAX input has index $\min\{j+h+2h, r\} = \min\{j+3h, r\}$; the MIN output (which certainly has index j) is fed into a comparator at level i whose MAX input has index $j + \max\{\lfloor \frac{h}{2} \rfloor, 1\}$. This completes the description of circuit $\mathcal{C}$.

We next show that circuit $\mathcal{C}$ has the claimed property. By the 0-1 principle, a circuit is a Δ-approximate-insertion circuit if it Δ-approximate-sorts every input sequence of the form 0^x1^y or 10^x1^y. (For example, 10^x1^y denotes the sequence $(1\,\underbrace{0,\ldots,0}_{x},\underbrace{1,\ldots,1}_{y})$.) Clearly, a standard circuit always sorts input sequences of the form 0^x1^y (which is already sorted). Hence, to show that $\mathcal{C}$ has the claimed property, we only need to prove that with probability at least $1-\rho^{\Theta(\log m)}$, a randomly generated faulty version of $\mathcal{C}$ outputs an $m^{\frac{6}{7}}$-approximate-sorted list on all input sequences of the form 10^x1^y. Moreover, since there are only $m-1$ sequences of the form 10^x1^y, for ρ sufficiently small, it suffices to show that on any fixed input sequence s of the form 10^x1^y, with probability at least $1-\rho^{\Theta(\log m)}$, a randomly generated faulty version of $\mathcal{C}$ produces an $m^{\frac{6}{7}}$-approximate-sorted list.

In what follows, we will focus on a particular input sequence s of the form 10^x1^y and a particular random faulty version, $\mathcal{C}'$, of $\mathcal{C}$. We need to show that with probability at least $1-\rho^{\Theta(\log m)}$, $\mathcal{C}'$ outputs an $m^{\frac{6}{7}}$-approximate-sorted list on input sequence s. We make a special mark on the 1 input to the top register, and we assume without loss of generality that the marked 1 will always be output to the MIN register when being compared with another 1. In addition, at each level, we mark the unique comparator that receives the marked 1 as one of its two input items. Clearly, the movement of the marked 1 is completely determined by the functionality of the marked comparators in $\mathcal{C}'$ (it has nothing to do with the functionality of other comparators). Let w be the index of the group that x belongs to. To prove that $\mathcal{C}'$ $m^{\frac{6}{7}}$-approximate-sorts s with high probability, it suffices to show that the marked 1 will be successfully moved to a register with index w with high probability.

When ρ is less than a sufficiently small constant, by a standard Chernoff bound argument, with probability at most $1-\rho^{\Theta(\log r)} = 1-\rho^{\Theta(\log m)}$, at most $\frac{\log r}{3}$ of the $5\log r$ marked comparators are faulty. In what follows, we will show that if at most $\frac{\log r}{3}$ marked comparators are faulty, then the marked 1 will be successfully moved to a register with index w. Given s and $\mathcal{C}'$, we observe how the marked 1 moves within $\mathcal{C}'$. For each $i = 1, 2, \ldots, 5\log r$, let d_i be the nonnegative integer such that the marked 1 is contained in a register with index $w - d_i$ immediately before level i; let h_i be the integer such that the other input register to the unique marked comparator at level i has index $w - d_i + h_i$. Note that according to our construction of $\mathcal{C}$, $h_i \geq 1$, for $i = 1, 2, \ldots, 5\log r$. We next prove that the marked 1 will be moved to the correct group of registers at the end of $\mathcal{C}'$ by showing that $d_{5\log r} = 0$. For any nonnegative integer k, we define $b(k)$ to be the number of bits in the binary representation of k

(assume $b(0) = 0$). Define a potential function F as follows:

$$F(i) = 3b(d_i) + |b(d_i) - b(h_i)|.$$

Claim 3.1.1 *(1) $F(i+1) \leq F(i) + 1$; (2) $F(i+1) \leq F(i) - 1$ when $d_i \geq 1$ and the marked comparator at level i is correct.*

By the inequalities $|x| - |y| \leq |x - y|$ and $|x + y| \leq |x| + |y|$,

$$F(i+1) - F(i) \leq 3\,(b(d_{i+1}) - b(d_i)) + |b(d_{i+1}) - b(d_i)| + |b(h_{i+1}) - b(h_i)|. \quad (3.1)$$

Hence, the first inequality in the claim follows from the fact that

$$b(d_{i+1}) \leq b(d_i) \quad (3.2)$$

and

$$b(h_{i+1}) \leq b(h_i) + 1. \quad (3.3)$$

Assuming that the marked comparator at level i is correct and that $d_i \geq 1$, we next prove

$$F(i+1) \leq F(i) - 1. \quad (3.4)$$

When $b(d_{i+1}) < b(d_i)$, inequalities 3.1 and 3.3 immediately imply inequality 3.4. Hence, by inequality 3.2, we only need to check inequality 3.4 under the assumption that

$$b(d_{i+1}) = b(d_i). \quad (3.5)$$

Given equality 3.5 and the assumption that the marked comparator at level i is correct, it must be that $b(h_i) \neq b(d_i)$. There are two cases to consider.

Case 1: $b(h_i) > b(d_i)$. In this case, the marked 1 will be output to the MIN output of the marked comparator at level i. Hence, $h_{i+1} = \max\{\lfloor \frac{h_i}{2} \rfloor, 1\} = \lfloor \frac{h_i}{2} \rfloor$ since

$b(h_i) > b(d_i) \geq 1$ implies $h_i \geq 2$. Hence,

$$b(h_{i+1}) = b(h_i) - 1.$$

This, together with the assumption of Case 1 and equality 3.5, implies that

$$F(i+1) - F(i) = (b(h_{i+1}) - b(d_{i+1})) - (b(h_i) - b(d_i)) = b(h_{i+1}) - b(h_i) = -1.$$

Case 2: $b(h_i) < b(d_i)$. In this case, the marked 1 will be output to the MAX output of the marked comparator at level i. Hence, either $h_{i+1} = 2h_i$ or the marked 1 will be compared against a register with the largest index, r, at level $i+1$. In the former case, we have

$$b(h_{i+1}) = b(h_i) + 1. \tag{3.6}$$

In the latter case, $h_{i+1} \geq d_{i+1}$, which implies $b(h_{i+1}) \geq b(d_{i+1}) = b(d_i) > b(h_i)$ (where we have used equality 3.5 and the assumption of Case 2). Again, inequality 3.6 holds (note that $b(h_{i+1}) \leq b(h_i) + 1$ clearly holds). Using equalities 3.5 and 3.6 and the assumption of Case 2, we have

$$F(i+1) - F(i) = (b(d_i) - b(h_i) - 1) - (b(d_i) - b(h_i)) = -1.$$

This proves inequality 3.4 and concludes the proof of Claim 3.1.1.

By Claim 3.1.1, we know that before d_i becomes 0, each correct marked comparator decreases the potential function by at least 1, and each faulty marked comparator increases the potential function by at most 1. Since we have assumed that at most $\frac{\log r}{3}$ among all the $5 \log r$ marked comparators are faulty, the potential decreases by at least $\frac{14}{3} \log r - \frac{1}{3} \log r = \frac{13}{3} \log r$ unless $d_i = 0$ for some $i \leq 5 \log r$. Since the initial potential is $F(0) \leq 3\lfloor \log r \rfloor + |\lfloor \log r \rfloor - 1| \leq 4 \log r$, this means that $F(5 \log r)$ must be negative unless $d_i = 0$ for some $i \leq 5 \log r$. Therefore, $d_i = 0$ for some $i \leq 5 \log r$, which means that at a certain level, the marked 1 is moved into a register with index w. Since we only have passive faults, once the marked 1 is moved into the register with index w, it will stay there until the end of C'. This proves that C has the claimed property.

By using an entirely symmetric construction, we can construct an $m^{\frac{6}{7}}$-approximate-insertion circuit that receives the unknown item at the bottom register. The same argument can be used to prove that the circuit does have the desired property.

To construct an approximate-insertion circuit that receives the unknown item at an arbitrary position, we use the following simple technique to increase the success probability of each comparator, as suggested in [26]. If we apply l consecutive comparators, each of which fails to work with probability upper bounded by ρ, to two registers, then the probability that the items contained in the two registers are unsorted after these l consecutive comparators is at most ρ^l. The l consecutive comparators can be viewed as a comparator whose failure probability is at most ρ^l. This technique will be used in many other passive-fault-tolerant circuit constructions, and will be referred to as the *replication* technique. For example, by using the replication technique, the approximate-insertion circuit that receives the unknown item at the top or bottom register can be made into an $O(l \log m)$-depth $(\rho, \rho^{\Theta(l \log m)})$-passive-fault-tolerant $m^{\frac{6}{7}}$-approximate-insertion circuit that receives the unknown item at the top or bottom register.

In general, when the unknown item has to be input to a given register r_x in a general position, we can construct the desired circuit by applying a circuit to all of the registers above and including r_x followed by another circuit to all of the registers below and including r_x. In one of the two circuits, we may have to use the replication technique to achieve the desired success probability if the number of inputs to that circuit is not large enough. ■

Lemma 3.1.2 *For any ρ less than a sufficiently small constant, there exists an explicit construction of an m-input $(\rho, \rho^{\Theta(\log m)})$-passive-fault-tolerant m^{α}-approximate-sorting circuit with $O(\log m)$ depth, where $\alpha < 1$ is a fixed constant.*

Proof: We construct the claimed circuit as follows. First, we apply a partial 1-AKS circuit to all m inputs. Let $X = S \bigcup X_1 \bigcup \ldots \bigcup X_{\bar{m}}$ be the partition of the output registers of the partial 1-AKS circuit as specified in Corollary 2.3.1, where $\bar{m} = \Theta(m^{1-\beta})$, $|S| = O(m^{\frac{3}{4}})$, and $|X_1| = |X_2| = \ldots = |X_{\bar{m}}| = \Theta(m^{\beta})$.

For $i \leq \bar{m}$ and $j \leq |X_1|$, let x_{ij} be the jth register in X_i, and let $Y_j = \{x_{ij} \mid 1 \leq i \leq \bar{m}\}$. According to Corollary 2.3.1, with probability at least $1 - \rho^{\Theta(\log m)}$, the item contained in x_{kj} is smaller than that contained in x_{lj} for any $k < l$, and Y_j thus contains a sorted list for $j \leq |X_1|$. For $j \leq |S|$, let s_j be the jth register in S. In parallel, we apply the $(\bar{m} + 1)$-input $\bar{m}^{\frac{6}{7}}$-approximate-insertion circuit of Lemma 3.1.1 to $\{s_j\} \bigcup Y_j$ for each $1 \leq j \leq |S|$. By Lemma 3.1.1, with probability at least $1 - |S| \; \rho^{\Theta(\bar{m})} = 1 - \rho^{\Theta(\log m)}$, all of the items in $\{s_j\} \bigcup Y_j$ are at most $\bar{m}^{\frac{6}{7}}$ away from the correct positions within $\{s_j\} \bigcup Y_j$ for all $j \leq |S|$. Since distance one within Y_j corresponds to distance at most $|X_1|$ within $\bigcup_{1 \leq j \leq |X_1|} Y_j = \bigcup_{1 \leq i \leq \bar{m}} X_i$, distance one within $\{s_j\} \bigcup Y_j$ corresponds to distance at most $|X_1| + |S| = \Theta(m^{\beta})$ in the whole circuit. Thus, distance $\bar{m}^{\frac{6}{7}}$ within $\{s_j\} \bigcup Y_j$ corresponds to distance of at most $\bar{m}^{\frac{6}{7}} \Theta(m^{\beta}) = \Theta(m^{(1-\beta)\frac{6}{7}+\beta}) \leq m^{\alpha}$ (where α can be any constant strictly less than 1 and strictly greater than $(1 - \beta)\frac{6}{7} + \beta$) in the whole circuit. ■

Proof of Theorem 3.1.1: To construct our passive-fault-tolerant sorting circuit with $O(\log n \log \log n)$ depth, we will repeatedly apply the approximate-sorting circuit of Lemma 3.1.2. In order to apply Lemma 3.1.2, we need to ensure that the failure probability for each comparator is upper bounded by a sufficiently small constant. This can be achieved by using the replication technique described on page 62. In particular, for any ρ and ϵ, we can construct a circuit that simulates a comparator whose failure probability is at most ϵ simply by replicating $\lceil \log_{\rho} \epsilon \rceil$ times a comparator whose failure probability is at most ρ. The circuit thus constructed will be referred to as a (ρ, ϵ)-*enforced* comparator hereafter. In order to apply Lemma 3.1.2, we need only set ϵ to a sufficiently small constant, which in turn leads to only constant factor replication. Then, we can build the whole circuit from these (ρ, ϵ)-enforced comparators. By applying this replication technique, the size and the depth of the whole circuit are both increased by only a constant factor. Ignoring this constant factor, we will assume that all of the comparators in our construction have a sufficiently small ρ and that Lemma 3.1.2 can be applied.

Let α be the constant in Lemma 3.1.2. Our circuit consists of $O(\log\log n)$ rounds. The ith round circuit has $O(\log n)$ depth, and it outputs a Δ_i-approximate-sorted list with probability at least $1 - \frac{1}{n^2}$ provided that the list produced by the $(i-1)$th round circuit is Δ_{i-1}-approximate-sorted, where Δ_i is a parameter determined by the following recurrence

$$\Delta_i = (4\Delta_{i-1})^{\alpha}, \qquad \text{for } i \geq 2, \tag{3.7}$$

with the boundary condition

$$\Delta_1 = n^{\alpha}. \tag{3.8}$$

The reason that we need to upper bound the failure probability of each round by $\frac{1}{n^2}$ instead of $\frac{1}{n}$ is that we will have $O(\log\log n)$ rounds and we will upper bound the overall failure probability by $O(\frac{1}{n^2}\log\log n) \leq \frac{1}{n}$ for n sufficiently large.

Recurrence 3.7 can be rewritten as

$$a\Delta_i = (a\Delta_{i-1})^{\alpha},$$

where $a = \left(\frac{1}{4}\right)^{\frac{\alpha}{1-\alpha}}$. Solving this recurrence, we find

$$\Delta_i = O(n^{\alpha^i}). \tag{3.9}$$

In the first round of our circuit, we apply the n^{α}-approximate-sorting circuit of Lemma 3.1.2 to all n inputs. In particular, we use the replication technique to make the failure probability of each comparator sufficiently small so that the term $\rho^{\Theta(\log n)}$ in Lemma 3.1.2 is smaller than $\frac{1}{n^2}$. Lemma 3.1.2 implies that, with probability at least $1 - \frac{1}{n^2}$, the outputs of the first round form a Δ_1-approximate-sorted list. The depth for the first round is $O(\log n)$.

For $i \geq 2$, we construct the ith round of the circuit so that with probability at least $1 - \frac{1}{n^2}$, the outputs of the ith round form a Δ_i-approximate-sorted list provided that the outputs from the $(i-1)$th round form a Δ_{i-1}-approximate-sorted list. At the beginning of the ith round, we group all the output registers from the $(i-1)$th round as follows. For $1 \leq k \leq \frac{n}{2\Delta_{i-1}}$, let X_k be the set of registers in positions $(k-1)2\Delta_{i-1}+1$

to $k2\Delta_{i-1}$. In parallel, we apply an approximate-sorting circuit of Lemma 3.1.2 to $X_1 \bigcup X_2$, another approximate-sorting circuit of Lemma 3.1.2 to $X_3 \bigcup X_4$, and so on. Then, in parallel, we apply an approximate-sorting circuit of Lemma 3.1.2 to $X_2 \bigcup X_3$, another approximate-sorting circuit of Lemma 3.1.2 to $X_4 \bigcup X_5$, and so on. We need the second set of approximate-sorting circuits to bring items across the boundaries of the first set of approximate-sorting circuits.

To analyze the behavior of the ith round circuit, we will make use the 0-1 principle. For any 0-1 sequence $s = (s_1, s_2, \ldots, s_m)$, we define the *dirty window* of s to be a subsequence of s of the form $(s_i, s_{i+1}, \ldots, s_j)$ such that (1) $s_i = 1$, (2) $s_j = 0$, (3) $s_k = 0$ for all $k < i$, and (4) $s_k = 1$ for all $k > j$. In other words, the dirty window of s is the shortest subsequence of s such that every item strictly before the subsequence is 0 and every item strictly after the subsequence is 1. Intuitively, the dirty window of a 0-1 sequence s is the part of s that we need to work on in order to sort s.

We next use the 0-1 principle to prove that if all of the constituent approximate-sorting circuits in the ith round work correctly as $(4\Delta_{i-1})^{\alpha}$-approximate-sorting circuits and if the list produced by the $(i-1)$th round is Δ_{i-1}-approximate-sorted, then the ith round circuit produces a Δ_i-approximate-sorted list. Suppose that we only have 0-1 inputs and that the list produced by the $(i-1)$th round is Δ_{i-1}-approximate-sorted. Then, the list input to the ith round contains a dirty window of size $2\Delta_{i-1}$ or less. This dirty window must be fully contained in one of the approximate-circuits involved in the ith round. (Recall that we have two sets of $(4\Delta_{i-1})$-input approximate-sorting circuits in the ith round, where the second set is offset from the first by $2\Delta_{i-1}$. Also note that passive faults cannot increase the size of the dirty window.) Hence, after the two sets of approximate-sorting circuits in the ith round, the output list is $(4\Delta_{i-1})^{\alpha}$-approximate-sorted, i.e., it is Δ_i-approximate-sorted.

In order to make sure that with sufficiently high probability, all of the constituent approximate-sorting circuits in the ith round work correctly, we need to do some careful calculations. In the above construction, if we simply apply the approximate-sorting circuits of Lemma 3.1.2 as in the first round, then the failure probability for each of the approximate-sorting circuits is $\frac{1}{(4\Delta_{i-1})^2} = \frac{1}{O(n^{2\alpha^{i-1}})}$, which is too large in comparison with the goal of $\frac{1}{n^2}$. To overcome this difficulty, we need to use the replication technique again. In particular, we construct the approximate-sorting circuits from (ρ, ϵ)-enforced comparators where ϵ satisfies

$$\epsilon^{\Theta(\log(4\Delta_{i-1}))} \leq \frac{1}{n^3}, \tag{3.10}$$

and where the constant behind the Θ-notation is the same as that in Lemma 3.1.2. By doing so, each of the approximate-sorting circuits fails with probability at most $\frac{1}{n^3}$. Hence, the probability that all of the constituent approximate-sorting circuits in the ith round are good is at least $1 - \frac{1}{n^2}$. To construct the (ρ, ϵ)-enforced comparators, we replicate each of the original comparators

$$\log_\rho \epsilon = O\left(\frac{3 \log_{\frac{1}{\rho}} n}{\log(4\Delta_{i-1})}\right) = O\left(\frac{\log_{\frac{1}{\rho}} n}{\log n^{\alpha^{i-1}}}\right)$$

times (where we have used equation 3.9). This replication results in an $O\left(\frac{\log_{\frac{1}{\rho}} n}{\log n^{\alpha^{i-1}}}\right)$ blowup in the original $O(\log(4\Delta_{i-1}))$-depth construction. Hence, the total depth of the ith round is

$$O\left(\log(4\Delta_{i-1})\right) O\left(\frac{\log_{\frac{1}{\rho}} n}{\log n^{\alpha^{i-1}}}\right) = O\left(\log n^{\alpha^{i-1}} \frac{\log_{\frac{1}{\rho}} n}{\log n^{\alpha^{i-1}}}\right) = O(\log_{\frac{1}{\rho}} n).$$

We repeatedly apply the above construction until every item is moved to within a constant number of positions of the correct position, i.e., we use i rounds such that $\Delta_i = O(1)$, which is equivalent to $n^{\alpha^i} = O(1)$ or $i = O(\log \log n)$ according to equation 3.9. Note that we should not apply the above construction all the way to the end to sort every item precisely to the correct position since we have only established the fault-tolerance properties of the l-AKS circuit for large m. When every item is within a constant number of positions of the correct position, we can combine the replication technique with any constant depth and constant size sorting circuit with a constant number of inputs to achieve exact sorting. This costs $O(\log n)$ additional depth. Overall, we have $O(\log \log n)$ rounds each of which has $O(\log n)$ depth and works with probability at least $1 - \frac{1}{n^2}$. Therefore, the total depth is $O(\log n \log\log n)$ and the failure probability is at most $O(\log\log n)\frac{1}{n^2} \leq \frac{1}{n}$ for n sufficiently large. ■

As we have pointed out in Section 1.3, the construction can be made $(\rho, \frac{1}{\text{poly}(n)})$-passive-fault-tolerant with only a constant factor increase in depth and size. In particular, the proof of Theorem 3.1.1 can easily be extended to establish the following corollary.

Corollary 3.1.1 *For any constants $\rho < 1$ and c, there exists an explicit construction of a $(\rho, \frac{1}{n^c})$-passive-fault-tolerant sorting circuit with $O(\log n \log\log n)$ depth.*

Theorem 3.1.2 *There exists an explicit construction of a passive-fault-tolerant selection circuit with asymptotically optimal size of $\Theta(n \log n)$.*

Proof: An $\Omega(n \log n)$ lower bound on the size of selection circuits was proved by Alekseyev [3] even in the fault-free case (see also Theorem A on pages 234-235 of [10]). In what follows, we give a passive-fault-tolerant construction with $O(n \log n)$ size.

Let α be the constant in Lemma 3.1.2, and let $\rho < 1$ be a constant upper bound on the failure probability of each comparator. Take a $(\rho, \frac{1}{n^2})$-passive-fault-tolerant n^α-approximate-sorting circuit $\mathcal{C}_1$ as in Lemma 3.1.2. Take another $(2n^\alpha + 2)$-input $(\rho, \frac{1}{n^2})$-passive-fault-tolerant-sorting circuit $\mathcal{C}_2$ as in Corollary 3.1.1, e.g., we can choose $c = \frac{3}{\alpha}$ in Corollary 3.1.1. Our passive-fault-tolerant selection circuit $\mathcal{C}$ consists of $\mathcal{C}_1$ followed by $\mathcal{C}_2$ with the middle $2n^\alpha + 2$ outputs of $\mathcal{C}_1$ being the inputs of $\mathcal{C}_2$. Clearly, the size of $\mathcal{C}$ is $O(n \log n) + O(n^\alpha \log n \log\log n) = O(n \log n)$.

We next show that the circuit constructed is a $(\rho, \frac{1}{n^2})$-passive-fault-tolerant circuit that outputs the median to the middle output register of $\mathcal{C}_2$. By the choice of $\mathcal{C}_1$ and $\mathcal{C}_2$, the probability that $\mathcal{C}_1$ is an n^α-approximate-sorting circuit and that $\mathcal{C}_2$ is a sorting circuit is at least $1 - \frac{2}{n^2} \geq 1 - \frac{1}{n}$. Hence, to show that $\mathcal{C}$ is a $(\rho, \frac{1}{n})$-passive-fault-tolerant selection circuit, we need only prove that when $\mathcal{C}_1$ is an n^α-approximate-sorting circuit and $\mathcal{C}_2$ is a sorting circuit, $\mathcal{C}$ always outputs the median input to its middle output register. When $\mathcal{C}_1$ is an n^α-approximate-sorting circuit, by the definition of approximate-sorting, the top $\frac{n}{2} - n^\alpha - 1$ outputs of $\mathcal{C}_1$ all contain items smaller than the median input to $\mathcal{C}$ and the bottom $\frac{n}{2} - n^{\alpha} - 1$ outputs of $\mathcal{C}_1$ all contain items larger than the median input to $\mathcal{C}$. Hence, the median input to $\mathcal{C}$ will be the median among all of the inputs to $\mathcal{C}_2$. When $\mathcal{C}_2$ is indeed a sorting circuit, the median input to $\mathcal{C}_2$, which is identical to the median input to $\mathcal{C}$, will be output to the middle output register of $\mathcal{C}_2$, which is the same as the middle output register of $\mathcal{C}$. ■

3.2 Reversal-Fault-Tolerant Circuits and Networks

In this section, we present our results for reversal faults. We consider both the circuit model and the network model. In Subsection 3.2.1, we use the fault-tolerance of the l-AKS circuit to construct a reversal-fault-tolerant $O(\log n)$-approximate-sorting circuit (for ρ less than a sufficiently small constant) with $O(n \log n (\log\log n)^2)$ size and $O(\log^2 n)$ depth. In addition, we present some general lower bounds for reversal-fault-tolerant approximate-sorting circuits. In Subsection 3.2.2, we use the approximate-sorting circuit of Subsection 3.2.1 to construct a reversal-fault-tolerant sorting network (for ρ less than a sufficiently small constant) with $O(n \log^{\log_2 3} n)$ size. This provides the first $o(n \log^2 n)$-size reversal-fault-tolerant sorting network, and it answers the open question posed by Assaf and Upfal [4]. In all the upper bound results of this section (except Lemma 3.2.6), we will need the assumption that ρ is less than a sufficiently small constant. Whether or not such an assumption is necessary is an open question. Note that no assumption will be made on ρ in the lower bound results.

3.2.1 Approximate-Sorting Circuits

In this subsection, we study approximate-sorting circuits that are tolerant to reversal faults. As defined on page 23, a Δ-approximate-sorting circuit is a circuit that outputs every item to within Δ of its correct position on all input permutations.

In Theorem 3.2.1 below, we show that any reversal-fault-tolerant Δ-approximate-sorting circuit has $\Delta = \Omega(\log n)$. This lower bound leads us to focus our attention on the study of reversal-fault-tolerant $O(\log n)$-approximate-sorting circuits in the remainder of the subsection. We continue in Theorem 3.2.2 with an $\Omega(\log^2 n)$ lower bound on the depth of any reversal-fault-tolerant $O(\log n)$-approximate-sorting circuit. The main result of this subsection is Theorem 3.2.3, where we construct a reversal-fault-tolerant $O(\log n)$-approximate-sorting circuit (for ρ less than a sufficiently small constant) with $O(n \log n (\log\log n)^2)$ size and asymptotically optimal depth of $O(\log^2 n)$. The size of this circuit is of particular importance when we further modify the circuit into a network with $o(n \log^2 n)$ size in the next subsection.

Theorem 3.2.1 *For any positive constant $\gamma < 1$, there exists a positive constant c depending on γ such that for any n-input circuit, where n is sufficiently large (depending on ρ and γ), with probability at least $1 - e^{-\frac{n^\gamma}{\log \frac{1}{\rho}}}$, on some input permutation, the circuit outputs some item to at least $c \log_{\frac{1}{\rho}} n$ away from the correct position.*

Proof: As pointed out in Section 1.3, to get the strongest possible result, ρ should be interpreted as the failure probability of each comparator in the circuit, as opposed to an upper bound on the failure probability. Throughout the proof, we further assume that $\rho \le \frac{1}{2}$. This assumption does not affect the generality of the proof, since a comparator failure probability of ρ can be viewed as failure probability of $1-\rho$ if the MIN/MAX output assignment is reversed. We first prove the following useful lemma.

Lemma 3.2.1 *For any m-input circuit C', where m is sufficiently large (depending on ρ), the probability that a randomly faulty version of C' is a Δ-approximate-sorting circuit is at most $1 - \rho^{2\Delta+1}$ for $\Delta < \frac{m-1}{2}$.*

The lemma states that in the reversal fault model, there is an inherent limitation on the success probability that can be achieved by any approximate-sorting circuit. The lemma also expresses a tradeoff between the accuracy of approximation and the degree of reliability that can be achieved by any circuit with reversal faults. Such a tradeoff will be used to prove Theorem 3.2.1.

For ease of notation, we next define the notion of a register segment. As described in Section 1.1, the comparators in a circuit are partitioned into disjoint levels. We assume that the comparators nearest to the inputs are at level 1. Also, we assume that the inputs to a circuit are provided at level 0 and the outputs are produced at level $d+1$, where d denotes the depth of the circuit. A *register segment* is defined to be the part of a register between two consecutive levels. Note that each level is used to partition all of the registers into corresponding register segments, even in those cases where the number of comparators in the level is less than half of the number of the registers. Hence, an m-input d-depth circuit contains $(d+1)m$ register segments, regardless of its size.

We next prove Lemma 3.2.1. Since the lemma is concerned with the functionality of $\mathcal{C}'$, and not the depth of $\mathcal{C}'$, we may assume without loss of generality that $\mathcal{C}'$ contains only one comparator at each level. From the top to the bottom, we label all of the registers in $\mathcal{C}'$ by $r_1, r_2, \ldots, r_m$. Let $r_{i,j}$ be the register segment of r_i between levels j and $j+1$ for $j = 0, 1, 2, \ldots, d$, where d denotes the depth of $\mathcal{C}'$. For $i = 1, 2, \ldots, m$ and $j = 0, 1, \ldots, d$, define $P_{i,j}$ to be the probability that 1 is contained in $r_{i,j}$ when a uniformly chosen permutation of $\{1, 2, \ldots, m\}$ is fed into a randomly faulty version of $\mathcal{C}'$.

For any index set $I \subset \{1, 2, \ldots, m\}$ such that $|I| < m$, consider the inequality

$$\sum_{i \in I} P_{i,j} \leq 1 - \rho^{|I|}. \tag{3.11}$$

It is easy to see that Lemma 3.2.1 follows from inequality 3.11 with $j = d$ and $|I| = 2\Delta + 1$. We now prove inequality 3.11 by induction on j.

Base case: $j = 0$. On a uniformly chosen input permutation of $\{1, 2, \ldots, m\}$, each input register segment contains 1 with probability $\frac{1}{m}$. Hence, we need to verify that $\frac{|I|}{m} \leq 1 - \rho^{|I|}$ for any I such that $|I| < m$. It suffices to show that

$$f(x) = 1 - \rho^x - \frac{x}{m} \geq 0 \tag{3.12}$$

for $0 \leq x \leq m-1$. Clearly, $f'(x) = -\rho^x \log_e \rho - \frac{1}{m}$ and $f'(x) = 0$ has a unique root in $[0, m-1]$, when m is large. Moreover, for m sufficiently large (depending on ρ), $f'(0) > 0$ and $f'(m-1) < 0$. Hence, inequality 3.12 follows from: (i) $f(0) = 0$ and

(ii) $f(m-1) = 1 - \rho^{m-1} - \frac{m-1}{m} = \frac{1}{m} - \rho^{m-1} > 0$ for m sufficiently large (depending on ρ).

Inductive step. Assuming that inequality 3.11 holds up to $j-1$, we next prove that inequality 3.11 holds for j. By our assumption on $\mathcal{C}'$, there is only one comparator at level j in $\mathcal{C}'$. Assume that this comparator connects two registers r_{i_1} and r_{i_2}. Clearly, if I contains both i_1 and i_2 or if I contains neither i_1 nor i_2, then $\sum_{i\in I} P_{i,j} = \sum_{i\in I} P_{i,j-1}$ and the correctness of inequality 3.11 for j follows from the induction hypothesis. Hence, we can assume without loss of generality that $i_1 \in I$ and $i_2 \notin I$. Let $I' = I - \{i_1\}$. By the definition of reversal faults and the fact that $\rho \le \frac{1}{2}$,

$$\begin{aligned}
\textstyle\sum_{i\in I} P_{i,j} &\le \textstyle\sum_{i\in I'} P_{i,j-1} + (1-\rho)\left(P_{i_1,j-1} + P_{i_2,j-1}\right) \\
&\le \textstyle\sum_{i\in I'} P_{i,j-1} + (1-\rho)\left(1 - \sum_{i\in I'} P_{i,j-1}\right) \\
&= \rho \textstyle\sum_{i\in I'} P_{i,j-1} + 1 - \rho \\
&\le \rho\left(1 - \rho^{|I'|}\right) + 1 - \rho \qquad \text{(by the induction hypothesis)} \\
&= 1 - \rho^{|I|}.
\end{aligned}$$

This concludes the inductive proof of inequality 3.11, as well as the proof of Lemma 3.2.1.

In what follows, let

$$m = \frac{1-\gamma}{3}\log_{\frac{1}{\rho}} n + 1, \tag{3.13}$$

and

$$\Delta = \left\lfloor \frac{m-2}{2} \right\rfloor. \tag{3.14}$$

We next consider an arbitrary n-input circuit $\mathcal{C}$, and prove the theorem by showing that the probability a randomly faulty version of $\mathcal{C}$ is a Δ-approximate-sorting circuit is at most $e^{-\frac{n^\gamma}{\log \frac{1}{\rho}}}$.

Roughly speaking, we will use Lemma 3.2.1 to prove Theorem 3.2.1 as follows. Consider an input sequence consisting of $\frac{n}{m}$ groups of size m such that all of the items in the ith group are smaller than all of the items in the jth group for $i < j$. If there is no comparator between items in different groups (intuitively, such comparisons provide no additional information), then each of the $\frac{n}{m}$ groups will be sorted by $\frac{n}{m}$

independent smaller circuits. By Lemma 3.2.1, the probability that all of these groups will be Δ-approximate-sorted is upper bounded by

$$\left(1-\rho^{2\Delta+1}\right)^{\frac{n}{m}} \le \left(1-\rho^{m-1}\right)^{\frac{n}{m}} = \left(1-n^{-\frac{1-\gamma}{3}}\right)^{\frac{n}{m}} \le \left(e^{-n^{-\frac{1-\gamma}{3}}}\right)^{\frac{n}{m}} \le e^{-\frac{n^{\gamma}}{\log\frac{1}{\rho}}}$$

for n sufficiently large depending on ρ and γ. To deal with the dependence problem caused by comparisons between items in different groups, it will be convenient to introduce some notations.

Suppose we are given a fault pattern F. (The definition of a fault pattern is given on page 10.) We use $\mathcal{C}(F)$ to denote the faulty version of $\mathcal{C}$ in which the (correct or faulty) behavior of each comparator is determined according to F. Let

$$\pi = \left(\underbrace{1,\ldots,1}_{m},\underbrace{2,\ldots,2}_{m},\ldots,\underbrace{\left\lceil\frac{n}{m}\right\rceil-1,\ldots,\left\lceil\frac{n}{m}\right\rceil-1}_{m},\underbrace{\left\lceil\frac{n}{m}\right\rceil,\ldots,\left\lceil\frac{n}{m}\right\rceil}_{n-m(\lceil\frac{n}{m}\rceil-1)}\right).$$

We define a comparator of $\mathcal{C}(F)$ to be a *crossing comparator* if it compares two distinct items when $\mathcal{C}(F)$ is executed on input sequence π. We use *Cross*(F) to denote the set of all crossing comparators of $\mathcal{C}(F)$. From the top to the bottom, we label the n registers by $r_1, r_2, \ldots, r_n$. We next describe a procedure to decompose $\mathcal{C}(F)$ into $\lceil\frac{n}{m}\rceil$ smaller circuits, $\mathcal{C}(F)_1, \ldots, \mathcal{C}(F)_{\lceil\frac{n}{m}\rceil}$. For $i = 1, \ldots, \lceil\frac{n}{m}\rceil$, we construct $\mathcal{C}(F)_i$ by applying the following 4-step procedure: (i) input π to $\mathcal{C}(F)$ and observe how the input items move within $\mathcal{C}(F)$; (ii) include all register segments (at all levels) of $\mathcal{C}$ that receive the item i; (iii) include all comparators for which both inputs receive the item i; (iv) for any crossing comparator that causes the item i to move from $r_{u,j}$ to $r_{v,j+1}$ for some j and $u \ne v$, directly connect $r_{u,j}$ and $r_{v,j+1}$ in $\mathcal{C}(F)_i$. Due to the direct connections introduced in step (iv), $\mathcal{C}(F)_i$ may contain some "twisted" registers, and look abnormal. However, these abnormalities will not prevent us from applying Lemma 3.2.1 to $\mathcal{C}(F)_i$. (As a matter of fact, we can construct a circuit that looks "normal" and that is equivalent to $\mathcal{C}(F)_i$, but we do not need this fact to apply Lemma 3.2.1.)

We next use a conditional probabilistic argument to show that for a random fault pattern F, with the probability claimed in Theorem 3.2.1, $\mathcal{C}(F)$ is not a Δ-approximate-sorting circuit. In particular, we will organize all of the fault patterns into groups and prove that for a randomly generated fault pattern F within each group, with the claimed probability, $\mathcal{C}(F)$ is not a Δ-approximate-sorting circuit.

We organize all of the fault patterns into groups according to *Cross*(F), as follows. For any two fault patterns F and F', we put F and F' in the same group if and only if *Cross*(F) = *Cross*(F'). Within each group thus constructed, we choose an arbitrary fault pattern as a representative of that group. Then, we list all of the representatives as $F_1, F_2, \ldots, F_t$ for some t. To prove the theorem, we only need to show that for each $s \le t$, for a randomly generated F such that *Cross*(F) = *Cross*(F_s), the probability that $\mathcal{C}(F)$ is a Δ-approximate-sorting circuit is at most $e^{\frac{-n^{\gamma}}{\log \frac{1}{\rho}}}$. The reason such a conditional probability is easier to analyze is as follows: By definitions, the information of F on *Cross*(F) completely determines the decomposition of $\mathcal{C}(F)$ into $\mathcal{C}(F)_1, \mathcal{C}(F)_2, \ldots, \mathcal{C}(F)_{\lceil \frac{n}{m} \rceil}$. Hence, full information of F on *Cross*(F) completely determines the constructions of $\mathcal{C}(F)_1, \mathcal{C}(F)_2, \ldots, \mathcal{C}(F)_{\lceil \frac{n}{m} \rceil}$ (although not the functionality of these circuits), and will allow us to apply Lemma 3.2.1 on the $\left\lfloor \frac{m}{n} \right\rfloor$ independent smaller circuits $\mathcal{C}(F)_1, \mathcal{C}(F)_2, \ldots, \mathcal{C}(F)_{\lfloor \frac{n}{m} \rfloor}$.

Now, for each $s \le t$, we have

$$\begin{aligned}
&\text{Prob}\left(\mathcal{C}(F) \text{ is a } \Delta\text{-approximate-sorting circuit} \mid \textit{Cross}(F) = \textit{Cross}(F_s)\right) \\
&\le \text{Prob}\left(\mathcal{C}(F)_i \text{ is a } \Delta\text{-approximate-sorting circuit } \forall i \le \left\lfloor \tfrac{n}{m} \right\rfloor \mid \textit{Cross}(F) = \textit{Cross}(F_s)\right) \\
&= \textstyle\prod_{1 \le i \le \lfloor \frac{n}{m} \rfloor} \text{Prob}\left(\mathcal{C}(F)_i \text{ is a } \Delta\text{-approximate-sorting circuit}\right) \\
&\qquad\qquad \text{(since the behaviors of } \mathcal{C}(F)_i\text{'s are mutually independent)} \\
&\le (1 - \rho^{m-1})^{\lfloor \frac{n}{m} \rfloor} \qquad \text{(by Lemma 3.2.1 and equation 3.14)} \\
&\le \left(1 - n^{-\frac{1-\gamma}{3}}\right)^{\lfloor \frac{n}{m} \rfloor} \qquad \text{(by equation 3.13)} \\
&\le \left(e^{-n^{-\frac{1-\gamma}{3}}}\right)^{\lfloor \frac{n}{m} \rfloor} \qquad \text{(by the inequality } 1 - x \le e^{-x}\text{)} \\
&\le e^{-\frac{n^{\gamma}}{\log \frac{1}{\rho}}} \qquad \text{(for } n \text{ sufficiently large, depending on } \rho \text{ and } \gamma \text{).}
\end{aligned}$$

This completes the proof of Theorem 3.2.1. ■

When ρ is a constant, the preceding theorem has the following immediate corollary, where the constant c depends on ρ.

Corollary 3.2.1 *There exists a constant c such that there exists no reversal-fault-tolerant $(c \log n)$-approximate-sorting circuit.*

Theorem 3.2.2 *For any positive constants c, ρ, and $\gamma < 1$, any (ρ, n^{-c})-reversal-fault-tolerant n^{γ}-approximate-sorting circuit has $\Omega(\log^2 n)$ depth.*

Proof: As pointed out in Section 1.3, to get the strongest possible result, ρ should be interpreted as the failure probability of each comparator in the circuit, as opposed to an upper bound on the failure probability. In fact, if ρ were interpreted as such an upper bound, the correctness of Theorem 3.2.2 would follow from Theorem 3.4.3 with $k = O(\log n)$.

In what follows, we assume

$$\epsilon < \frac{1}{2}. \tag{3.15}$$

Let

$$\lambda = \left\lceil \log_4 \left(\frac{n-1}{2\Delta} \right) \right\rceil - 1, \tag{3.16}$$

and

$$\mu = \left\lfloor \log_\rho 2\epsilon \right\rfloor . \tag{3.17}$$

The correctness of Theorem 3.2.1 follows from the next lemma with $\epsilon = n^{-c}$ and $\Delta = n^{\gamma}$.

Lemma 3.2.2 *Any (ρ, ϵ)-reversal-fault-tolerant Δ-approximate-insertion circuit has depth strictly greater than $\lambda\mu$.*

We now prove Lemma 3.2.2. Consider an arbitrary circuit C that is a (ρ, ϵ)-reversal–fault-tolerant Δ-approximate-insertion circuit, and apply one of the two input permutations $(1, 2, \ldots, n)$ and $(n, 1, 2, \ldots, n-1)$, each with probability $\frac{1}{2}$, to a randomly faulty version of C. In such a setting, the item contained in each register segment is a random variable depending on the random input permutation and the random faults assigned to C. (Recall that, as defined on page 70, a register segment is the part of a register between two consecutive levels in the circuit.)

For each register segment x, let X denote the random variable corresponding to the item received by that register segment, and let *label*(x) in $[1,n]$ be chosen to satisfy

$$\mathrm{Prob}(X \geq \mathit{label}(x)) \geq \frac{1}{2}$$

and

$$\mathrm{Prob}(X \leq \mathit{label}(x)) \geq \frac{1}{2}.$$

Note that *label*(x) always exists but may not be unique.

Define a register segment y to be a *descendant* of a register segment x if the item received by x could subsequently enter y in some faulty or non-faulty version of C. Also, define the *descendant set* $D(x,i)$ of x to be the set of all descendants of x between levels i and $i+1$.

For any i such that $0 \leq i \leq \lambda$, and for any pair of numbers, r_1 and r_2, such that

$$r_2 - r_1 \geq \frac{n-1}{4^i} > 2\Delta,$$

we make the following pair of definitions. First, a register segment x between levels $i\mu$ and $i\mu + 1$ is defined to be (r_1, r_2)-*bad* if and only if x receives an item less than or equal to r_1 with probability at least ϵ, and receives an item greater than or equal to r_2 with probability at least ϵ. Second, a descendant set $D(x, i\mu)$ is defined to be (r_1, r_2)-*bad* if and only if one of the following two conditions holds:

(i) x receives an item less than or equal to r_1 with probability at least ϵ, and the label of every register segment in $D(x, i\mu)$ is greater than or equal to r_2, or

(ii) x receives an item greater than or equal to r_2 with probability at least ϵ, and the label of every register segment in $D(x, i\mu)$ is less than or equal to r_1.

Claim 3.2.1 *No output register segment of C can be (r_1, r_2)-bad. Furthermore, if C*

has depth $\lambda\mu$, then C cannot contain an (r_1, r_2)-bad descendant set $D(x, \lambda\mu)$ for any register segment x.

The first part of the claim is straightforward since C is a (ρ, ϵ)-reversal-fault-tolerant Δ-approximate-insertion circuit. Now consider the second part of the claim. Assume for the purposes of contradiction that C has depth of $\lambda\mu$ and contains an (r_1, r_2)-bad descendant set $D(x, \lambda\mu)$ for some x. Without loss of generality, we can assume that the first condition in the definition of a descendant set holds (the other case can be handled by an entirely symmetric argument). Under this assumption, with probability at least ϵ, x receives an item less than or equal to r_1. This item will eventually enter some register segment in $D(x, \lambda\mu)$. On the other hand, every register segment in $D(x, \lambda\mu)$ receives an item greater than or equal to r_2 with probability at least $\frac{1}{2} > \epsilon$. Hence, as required by the insertion problem, every register segment y in $D(x, \lambda\mu)$ is assigned an output rank greater than or equal to $r_2 - \Delta$. But if any register segment in $D(x, \lambda\mu)$ receives an item less than or equal to r_1, then C will fail since $(r_2 - \Delta) - r_1 > \Delta$. This proves the claim.

Claim 3.2.2 *For any i such that $0 \leq i \leq \lambda$, the circuit C contains either an (r_1, r_2)-bad register segment between levels $i\mu$ and $i\mu + 1$ or an (r_1, r_2)-bad descendant set $D(x, i\mu)$ for some register segment x.*

We prove the claim by induction on i. For the base case, $i = 0$, note that the top input register segment x receives 1 with probability $\frac{1}{2} > \epsilon$, and n with probability $\frac{1}{2} > \epsilon$. Thus, x is $(1, n)$-bad.

Now assume that the lemma holds for $i = j$, $0 \leq j < \lambda$, and consider the induction step, $i = j + 1$. Let $s = \frac{n-1}{4^j}$.

We first argue that if there is a register segment x between levels $j\mu$ and $j\mu + 1$, and a register segment y belonging to $D(x, (j+1)\mu)$, such that $|\mathit{label}(y) - \mathit{label}(x)| \geq \frac{s}{4}$, then y is either $(\mathit{label}(x), \mathit{label}(y))$-bad or $(\mathit{label}(y), \mathit{label}(x))$-bad. To see this, assume without loss of generality that $\mathit{label}(y) - \mathit{label}(x) \geq \frac{s}{4}$ (the case where $\mathit{label}(x) - \mathit{label}(y) \geq \frac{s}{4}$ can be handled by an entirely symmetric argument). Note that y receives an item less than or equal to $\mathit{label}(x)$ with probability at least $\frac{\rho^\mu}{2}$, which is at least ϵ

by equation 3.17. Furthermore, y receives an item greater than or equal to *label*(y) with probability at least $\frac{1}{2} > \epsilon$. Hence, y is (*label*(x), *label*(y))-bad, since

$$\mathit{label}(y) - \mathit{label}(x) \geq \frac{s}{4} = \frac{n-1}{4^{j+1}} \geq \frac{n-1}{4^{\lambda}} > 2\Delta \tag{3.18}$$

by equation 3.16.

By the argument of the preceding paragraph, we now have the additional assumption that for every register segment x between levels $j\mu$ and $j\mu+1$, and every register segment y belonging to $D(x,(j+1)\mu)$, $|\mathit{label}(y) - \mathit{label}(x)| < \frac{s}{4}$. Proceeding with the induction step, there are two cases to consider.

Case 1: There is an (r_1, r_2)-bad register segment x at level $j\mu$. Let S denote the set of labels associated with the register segments in $D(x,(j+1)\mu)$. By our additional assumption, every pair of labels in S differ by strictly less than $\frac{s}{2}$. On the other hand, $r_2 - r_1 \geq \frac{n-1}{4^j} = s$. Hence, either every label in S exceeds r_1 by at least $\frac{s}{4}$, or r_2 exceeds every label in S by at least $\frac{s}{4}$. Therefore, $D(x,(j+1)\mu)$ is either $(r_1, r_1 + \frac{s}{4})$-bad or $(r_2 - \frac{s}{4}, r_2)$-bad since $\frac{s}{4} > 2\Delta$ by equation 3.18.

Case 2: There is an (r_1, r_2)-bad descendant set $D(x, j\mu)$. We will assume that the first condition in the definition of an (r_1, r_2)-bad descendant set is satisfied; the other case can be handled by an entirely symmetric argument. Each register segment z in $D(x,(j+1)\mu)$ is a descendant of some register segment y in $D(x, j\mu)$. By our additional assumption, $\mathit{label}(y) - \mathit{label}(z) < \frac{s}{4}$. Furthermore, by the first condition in the definition of an (r_1, r_2)-bad descendant set, $\mathit{label}(y) \geq r_2$. Hence, $\mathit{label}(z) \geq r_2 - \frac{s}{4}$, and $D(x,(j+1)\mu)$ is $(r_1, r_2 - \frac{s}{4})$-bad. This concludes the inductive step for the proof of Claim 3.2.2.

Finally, it is immediate that any circuit of depth d (not necessarily a multiple of μ) can be viewed as a circuit of depth $\left\lceil \frac{d}{\mu} \right\rceil \mu$ with the same behavior. Hence, the correctness of Lemma 3.2.2 follows from Claims 3.2.1 and 3.2.2. This completes the proof of Theorem 3.2.2. ■

We remark that, as indicated in the proof, the $\Omega(\log^2 n)$ depth lower bound in Theorem 3.2.2 actually holds for the much simplier problem of n^{γ}-approximate-insertion.

Theorem 3.2.3 *For any ρ less than a sufficiently small constant, there exists an explicit construction of a reversal-fault-tolerant $O(\log n)$-approximate-sorting circuit with $\Theta(\log^2 n)$ depth and $O(n \log n (\log\log n)^2)$ size.*

To prove Theorem 3.2.3, we first prove a few useful lemmas.

Lemma 3.2.3 *For any ρ less than a sufficiently small constant and for $l \geq \log \frac{m}{l}$, there exists an explicit construction of an m-input $(\rho, \rho^{\Theta(l)})$-reversal-fault-tolerant l-approximate-insertion circuit with $O(l \log \frac{m}{l})$ depth and $O(ml)$ size.*

Proof: Without loss of generality, we assume that both m and l are integral powers of 2. From the top to the bottom, we label all the registers by $r_1, \ldots, r_m$, and we assume that the unique unknown item is input to a particular register r_x. For $1 \leq i \leq \log \frac{m}{l}$, let $R_i = \{r_j \mid j \equiv x \bmod \frac{m}{2^i l}\}$. Our circuit consists of $\log \frac{m}{l}$ rounds. In the ith round, we group all the registers in R_i from the top to the bottom into groups $X_{i,1}, X_{i,2}, \ldots, X_{i,2^i}$ so that each of the groups contains l registers. First, in parallel, we apply a set of $2l$-input and $4l$-depth odd-even transposition circuits to $X_{i,1} \bigcup X_{i,2}$, $X_{i,3} \bigcup X_{i,4}$, and so on. Second, in parallel, we apply another set of $2l$-input and $4l$-depth odd-even transposition circuits to $X_{i,2} \bigcup X_{i,3}$, $X_{i,4} \bigcup X_{i,5}$, and so on.

Since each R_i contains 2^i groups of the form $X_{i,j}$, the total number of odd-even transposition circuits in the ith round is at most

$$2 \cdot 2^{i-1} = 2^i.$$

Hence, the total number of odd-even transposition circuits used in the entire circuit is at most

$$\sum_{1 \leq i \leq \log \frac{m}{l}} 2^i \leq 2^{1+\log \frac{m}{l}} = \frac{2m}{l}. \tag{3.19}$$

Since each of the odd-even transposition circuits has size of $O(l^2)$, the size of the entire circuit is $O(\frac{m}{l} l^2) = O(ml)$. Also, since we have $\log \frac{m}{l}$ rounds each of which consists of two sets of $O(l)$-depth odd-even transposition circuits, the depth of the entire circuit is $O(l \log \frac{m}{l})$.

By Lemma 2.3.1 with $\vartheta = \frac{1}{32}$, each of the odd-even transposition circuits used in the entire circuit is an $\frac{l}{32}$-approximate-sorting circuit with probability at least $1 - \rho^{\Theta(l)} = 1 - \rho^{\Theta(l)}$. Hence, by inequality 3.19, the probability that all of the constituent odd-even transposition circuits are $\frac{l}{32}$-approximate-sorting circuits is at least

$$1 - O\left(\frac{m}{l}\right) \rho^{\Theta(l)} = 1 - \rho^{\Theta(l)},$$

where we have used the fact that ρ is sufficiently small and $l \geq \log \frac{m}{l}$.

Thus, to prove the lemma, it remains to show that the circuit thus constructed is an l-approximate-insertion circuit when all of the constituent odd-even transposition circuits are $\frac{l}{32}$-approximate-sorting circuits. Assuming that all of the constituent odd-even transposition circuits are $\frac{l}{32}$-approximate-sorting circuits, we prove by induction on i that the items of R_i form an $\frac{l}{8}$-approximate-sorted list after the ith round. (This suffices to prove that our circuit is an l-approximate-insertion circuit since $R_{\log \frac{m}{l}}$ contains all of the m registers.)

The base case $i = 1$ is trivial, since the first round actually consists of a single $\frac{l}{32}$-approximate-sorting circuit. Assuming that the items from the $(i-1)$th round form an $\frac{l}{8}$-approximate-sorted list, we show that the items from the ith round form an $\frac{l}{8}$-approximate-sorted list. By the 0-1 principle, we need only consider the case where the inputs are 0s or 1s. Recall that, as defined on page 65, the dirty window of a 0-1 sequence s is the shortest subsequence of s such that every item strictly before the subsequence is 0 and every item strictly after the subsequence is 1. Since the sequence from the $(i-1)$th round is assumed to be $\frac{l}{8}$-approximate-sorted, the input sequence to the ith round is clearly an $\frac{l}{4}$-approximate-sorted list and has a dirty window of size $\frac{l}{2}$ or less. There are two cases to consider.

Case 1: The dirty window of the sequence input to the ith round is fully contained in a circuit that belongs to the first set of $\frac{l}{32}$-approximate-sorting circuits in the ith round. In this case, the size of the dirty window will be decreased to $\frac{2l}{32}$ or less by the first set of $\frac{l}{32}$-approximate-sorting circuits in the ith round. Then, the second set of $\frac{l}{32}$-approximate-sorting circuits may increase the size of the dirty window by at most an additive term of $\frac{2l}{32}$. (Note that this is different from the case of passive faults, which cannot increase the size of a dirty window.) Hence, the final output sequence from the ith round has a dirty window of size at most $\frac{4}{32}l = \frac{l}{8}$, and is $\frac{l}{8}$-approximate-sorted.

Case 2: The dirty window of the sequence input to the ith round is *not* fully contained in any of the circuits in the first set of $\frac{l}{32}$-approximate-sorting circuits in the ith round. In this case, the first set of $\frac{l}{32}$-approximate-sorting circuits in the ith round may increase the size of the dirty window by an additive term of $\frac{2l}{32}$. Hence, the sequence input to the second set of $\frac{l}{32}$-approximate-sorting circuits has a dirty window of size at most $\frac{l}{2} + \frac{2l}{32} = \frac{9l}{16}$. By the assumption of Case 2 and the fact that the boundaries of the first set of $\frac{l}{32}$-approximate-sorting circuits are the centers of the second set of $2l$-input $\frac{l}{32}$-approximate-sorting circuits, this dirty window of size $\frac{9l}{16}$ or less is fully contained in a circuit that belongs to the second set of $\frac{l}{32}$-approximate-sorting circuits. Hence, the final output sequence produced by the second set of $\frac{l}{32}$-approximate-sorting circuits in the ith round has a dirty window of size at most $\frac{2l}{32}$, and is thus $\frac{l}{8}$-approximate-sorted. ■

Lemma 3.2.4 *For $l \le m^{\frac{1-\beta}{2}}$ (where β is the constant specified in Corollary 2.3.1) and for any ρ less than a sufficiently small constant, there is a constant $\alpha < 1$ such that there exists an explicit construction of an m-input $(\rho, \rho^{\Theta(l \log m)})$-reversal-fault-tolerant m^α-approximate-sorting circuit with $O(l \log^2 m)$ depth and $O(ml \log m)$ size.*

Proof: We use a similar technique as in the proof of Lemma 3.1.2. First, we apply the partial l-AKS circuit of Theorem 2.3.1. Then, we apply a set of $(\rho, \rho^{\Theta(l \log m)})$-reversal-fault-tolerant $(l \log m)$-approximate-insertion circuits of Lemma 3.2.3 in a fashion similar to that in the proof of Lemma 3.1.2. The assumption $l \le m^{\frac{1-\beta}{2}}$ implies $l \le m^{\frac{1}{8}}$, which makes it possible to apply Corollary 2.3.1 in the first step. In the second step, we can argue that every item is output to within $O(l \log m)\Theta(m^\beta)$ of the correct position. The assumption $l \le m^{\frac{1-\beta}{2}}$ guarantees that every output item is within $O(l \log m)\Theta(m^\beta) < m^\alpha$ of the correct position for any $\alpha > \frac{1+\beta}{2}$. In particular, the constant α in this lemma is different from that of Lemma 3.1.2.

By Theorem 2.3.1 and Lemma 3.2.3, the depth of the entire circuit is $O(l \log m) + O(l \log m \log \frac{m}{l \log m}) = O(l \log^2 m)$, and the size of the entire circuit $O(ml \log m) + O(ml \log m) = O(ml \log m)$. ■

Lemma 3.2.5 *For any ρ less than a sufficiently small constant and $l \geq c \log m$, where c is a sufficiently large constant, there exists an explicit construction of an m-input $(\rho, \rho^{\Theta(l)})$-reversal-fault-tolerant l-approximate-sorting circuit with $O(l \log^2 m)$ depth and $O(ml \log^2 m)$ size.*

Proof: Our construction is similar to Batcher's odd-even merge sort. We first construct a $(\rho, \rho^{\Theta(l)})$-reversal-fault-tolerant circuit that l-approximate-merges two l-approximate-sorted lists. Given two t-item lists L_1 and L_2, define $L_{1\alpha}$ and $L_{2\alpha}$ to consist of the even-index elements of L_1 and L_2 (resp.), and define $L_{1\beta}$ and $L_{2\beta}$ to consist of odd-index elements of L_1 and L_2 (resp.). If L_1 and L_2 are l-approximate-sorted, then so are $L_{1\alpha}$, $L_{2\alpha}$, $L_{1\beta}$, and $L_{2\beta}$. Next, recursively l-approximate-merge $L_{1\alpha}$ with $L_{2\beta}$ to form L_α, and $L_{1\beta}$ with $L_{2\alpha}$ to form L_β. By definition, L_α and L_β are l-approximate-sorted. Next, shuffle L_α with L_β to form a list L' that is $2l$-approximate-sorted. Finally, partition L' into contiguous blocks consisting of $16l$ items each, and apply a $16l$-input and $32l$-depth odd-even transposition circuit as described in Lemma 2.3.1 (with $\vartheta = \frac{1}{32}$) to each block. Finish up by repartitioning the resulting list into contiguous blocks of size $16l$ so that the boundaries of the earlier blocks fall in the centers of the current blocks, and then applying a $16l$-input and $32l$-depth odd-even transposition circuit to each of the blocks.

By an argument similar to that for Lemma 3.2.3, we can show that if all of the odd-even transposition circuits involved in the entire approximate-merging circuit work correctly as $\frac{l}{4}$-approximate-sorting circuits, then the entire circuit works correctly as an l-approximate-merging circuit. By Lemma 2.3.1 with $\vartheta = \frac{1}{32}$, the probability that a particular odd-even transposition circuit is not an $\frac{l}{4}$-approximate-sorting circuit is at most $\rho^{\Theta(l)}$. On the other hand, the total number of odd-even transposition circuits involved in the entire circuit is upper bounded by a polynomial in m. Hence, the probability that all of the odd-even transposition circuits function correctly as

$\frac{l}{4}$-approximate-sorting circuits is at least $1 - \text{poly}(m) \cdot \rho^{\Theta(l)} = 1 - \rho^{\Theta(l)}$ when ρ is sufficiently small and $l \geq c \log m$ for a sufficiently large constant c.

The depth of the l-approximate-merging circuit thus constructed is determined by the following recurrence

$$M(t) = M\left(\frac{t}{2}\right) + O(l)$$

with the boundary condition $M(t) = \Theta(1)$ for $1 \leq t \leq l$, where $M(t)$ stands for the depth of a circuit for l-approximate-merging two t-item lists. Solving the recurrence, we find that our circuit has depth of $M(t) = O(l \log \frac{t}{l})$.

Given the approximate-merging circuit thus constructed, we can construct the claimed l-approximate-sorting circuit by partitioning the m items to be sorted into two subset of $\frac{m}{2}$ items each, l-approximate-sorting the subsets recursively, and then l-approximate-merging them. This technique leads to the following recurrence for the depth of the l-approximate-sorting circuit

$$D(m) = D\left(\frac{m}{2}\right) + M\left(\frac{m}{2}\right) = D\left(\frac{m}{2}\right) + O\left(l \log \frac{m}{l}\right),$$

with the boundary condition $D(m) = \Theta(1)$ for $1 \leq m \leq l$. Solving the recurrence, we find that $D(m) = O(l \log^2 \frac{m}{l})$. Since each level of the circuit contains at most $\frac{m}{2}$ comparators, the size of the circuit is $O(ml \log^2 \frac{m}{l})$. ■

Proof of Theorem 3.2.3: We construct the circuit in a fashion similar to that of Theorem 3.1.1. In $O(\log \log n)$ rounds, we repeatedly apply the approximate-sorting circuit of Lemma 3.2.4 with some appropriately chosen value of l. The ith round consists of two sets of approximate-sorting circuits where the boundaries of the second set of circuits fall in the centers of the first set of circuits. Our hope is that with probability at least $1 - \frac{1}{n^2}$, the ith round outputs every item to within $O(n^{\alpha^i})$ of the correct position (where α is the constant specified in Lemma 3.2.4) provided that each of the outputs from the $(i-1)$th round is $O(n^{\alpha^{i-1}})$ within the correct position. The detailed parameter choices on how to apply the approximate-sorting circuits in each of the rounds are similar to those in the proofs of Theorem 3.1.1 and Lemma 3.2.3, and the detailed argument in Lemmas 3.2.3 can be applied to show that our circuit does have the desired property provided that all of the constituent approximate-sorting circuits function correctly.

In the proof of Theorem 3.1.1, we have used the replication technique to guarantee a success probability of at least $1-\frac{1}{n^2}$ in each round. For reversal faults, however, if we replicate a comparator many times, then the outcome will be completely determined by the behavior of the last comparator. Thus, we need a new method to replace the replication technique. Lemma 2.3.1 will essentially play a role similar to that of the replication technique. Note that we have assumed ρ to be sufficiently small in both Lemma 2.3.1 and Theorem 2.3.1. Hence, we can prove the current theorem only under the assumption that ρ is less than a sufficiently small constant.

Returning to the construction of the reversal-fault-tolerant $O(\log n)$-approximate-sorting circuit, let l_i be the parameter choice of l in Lemma 3.2.4 for the ith round, in which we need circuits with $\Theta(n^{\alpha^{i-1}})$ items each. We will choose $l_i = \Theta(\frac{1}{\alpha^{i-1}\log\frac{1}{\rho}})$ so that

$$\Theta(l_i \log n^{\alpha^{i-1}}) = \log_{\frac{1}{\rho}} n^3, \tag{3.20}$$

where the constant behind the Θ-notation is the same as that of Lemma 3.2.4. This guarantees that the failure probability for the ith round is at most $n\rho^{\Theta(l_i \log n^{\alpha^{i-1}})} \leq \frac{1}{n^2}$. By Lemma 3.2.4 and equation 3.20, the depth and the size for the ith round are

$$O\left(l_i \log^2 n^{\alpha^{i-1}}\right) = O\left(\log n \log n^{\alpha^{i-1}}\right) \tag{3.21}$$

and

$$O\left(nl_i \log n^{\alpha^{i-1}}\right) = O\left(n \log n\right), \tag{3.22}$$

respectively. In order to apply Lemma 3.2.4, we need to stop this procedure immediately before round i if i satisfies the following equation:

$$l_i = \Theta\left(\left(n^{\alpha^{i-1}}\right)^{\frac{1-\beta}{2}}\right), \tag{3.23}$$

where β is the constant specified in Corollary 2.3.1. Let i_0 be the smallest i that satisfies equation 3.23. Simple calculations show that $l_{i_0} = \Theta(\log n)$, $n^{\alpha^{i_0-1}} = \Theta(\log^{\frac{2}{1-\beta}} n)$, and $i_0 = \Theta(\log\log n)$. This means that every item has been moved to within $O(n^{\alpha^{i_0-1}}) = O(\log^{\frac{2}{1-\beta}} n)$ of the correct position at the end of the $(i_0 - 1)$th round.

Given the $O(\log^{\frac{2}{1-\beta}} n)$-approximate-sorted list, we apply the $O(\log n)$-approximate-sorting circuits of Lemma 3.2.5 with $l = O(\log n)$ to contiguous blocks of $O(\log^{\frac{2}{1-\beta}} n)$ items twice where the boundaries of the second set of circuits fall in the centers of the first set of circuits. (Detailed parameter choices are similar to those in the proof of Lemma 3.2.3.) The depth and size of this final round are $O(\log n(\log\log n)^2)$ and $O(n \log n(\log\log n)^2)$, respectively.

Now, by expressions 3.21 and 3.21, the depth the entire circuit is

$$O\left(\log n(\log\log n)^2\right) + \sum_{1\leq i<i_0} O\left(\log n \log n^{\alpha^{i-1}}\right) = O\left(\log^2 n\right),$$

and the size of the entire circuit is

$$O\left(n\log n(\log\log n)^2\right) + \sum_{1\leq i<i_0} O\left(n\log n\right) = O\left(n\log n(\log\log n)^2\right).$$

■

If we are only interested in a circuit with $O(\log^2 n)$ depth and do not particularly care about its size, there is a much simpler construction that does not depend on Theorem 2.3.1. In fact, the proof technique of Lemma 2.3.2 enables us to construct an m-input $(\rho, \frac{1}{n^2})$-reversal-fault-tolerant ϵ-halver with $O(\log n)$ depth for any positive constant ϵ and $m = \Omega(\log n)$. Replacing each of the ϵ-halvers in the AKS circuit by a reversal-fault-tolerant ϵ-halver constructed in this manner, we can get a reversal-fault-tolerant $O(\log n)$-approximate-sorting circuit with $O(\log^2 n)$ depth relatively easily. Since the fault-tolerance of the l-AKS circuit is very difficult to prove, the existence of the simpler construction with the asymptotically optimal depth is worth mentioning. Nevertheless, we need the fault-tolerance of the l-AKS circuit to achieve the better size bound, which is crucial for the next subsection.

3.2.2 Sorting Networks

In this subsection, we use the reversal-fault-tolerant $O(\log n)$-approximate-sorting circuit designed in the preceding subsection to construct a reversal-fault-tolerant sorting network of $O(n \log^{\log_2 3} n)$ size (for ρ less than a sufficiently small constant). This is the first reversal-fault-tolerant sorting network of $o(n \log^2 n)$ size, and it answers the open question posed by Assaf and Upfal [4]. As in [4], we will assume that the replicators are fault-free. This is not a particularly unreasonable assumption since replicators can be hard-wired and they do not contain any logic elements.

Theorem 3.2.4 *For any ρ less than a sufficiently small constant, there exists an explicit construction of a reversal-fault-tolerant sorting network with $O(n \log^{\log_2 3} n)$ size.*

The next lemma addresses how to use a network to compute a certain majority-like function. (As for the sorting problem, the network consists of comparators, which is subject to reversal faults, and replicators; but unlike in a sorting network, the computation result is output to only one of the many registers in the network.) In particular, we are interested in a network that computes the majority function correctly provided that a large fraction (much larger than $\frac{1}{2}$) of the inputs are all 0s or all 1s. Formally, for any constant $r \in (\frac{1}{2}, 1)$, we define an *r-MAJORITY function* with n inputs to be a Boolean function that outputs 1 if more than rn of the inputs are all 1s and that outputs 0 if more than rn of the inputs are all 0s. (We do not care how the function behaves if neither the number of 0s nor the number of 1s exceeds rn.)

Lemma 3.2.6 ([9]) *For some constant $r \in (\frac{1}{2}, 1)$ and for any constant $\rho < \frac{1}{2}$, there exists an explicit construction of a (ρ, ϵ)-reversal-fault-tolerant network with $O\left((\log_\rho \epsilon)^{\log_2 3}\right)$ size that computes the r-MAJORITY function of $O(\log_\rho \epsilon)$ inputs.*

Proof: See [9]. ∎

Proof of Theorem 3.2.4: Our network consists of two parts. The first part is a reversal-fault-tolerant $O(\log n)$-approximate-sorting circuit described in Theorem 3.2.3. This part has size of $O(n \log n (\log n \log n)^2)$. (We actually need the circuit to be $(\rho, \frac{1}{2n})$-fault-tolerant instead of $(\rho, \frac{1}{n})$-fault-tolerant. This requirement can be satisfied since the circuit of Theorem 3.2.3 is actually $(\rho, O(\frac{\log^2 n}{n^2}))$-fault-tolerant.) At the end of this part, with probability at least $1 - \frac{1}{2n}$, every item is within $O(\log n)$ of the correct position. Note that we do not need any replicators nor more than n registers in this part.

The second part of our network moves all of the items to the correct positions with probability at least $1 - \frac{1}{2n}$, provided that each of the inputs to this part is at most $O(\log n)$ away from the correct position. To construct the second part, it suffices to construct an $O(\log n)$-input $(\rho, \frac{1}{n^2})$-reversal-fault-tolerant sorting network with $O(\log n \log^{\log_2 3} n)$ size, since we can finish the second part by applying such networks to contiguous blocks of $O(\log n)$ items twice, where the boundaries of the second set of networks fall in the centers of the first set of networks.

Such an $O(\log n)$-input $(\rho, \frac{1}{n^2})$-reversal-fault-tolerant sorting network can be constructed by using the Assaf-Upfal method [4] with some modifications. Let $\mathcal{C}$ be an $O(\log n)$-input sorting circuit with $O(\log \log n)$ depth (e.g., let $\mathcal{C}$ be an AKS circuit with $O(\log n)$ inputs). Replace each original register r_i by a block of registers $R_i = \{r_{i1}, r_{i2}, \ldots, r_{is}\}$ where $s = O(\log n)$. Also, replace each comparator between r_i and r_j by s comparators that connect r_{ik} and r_{jk} for each $k \leq s$. After each set of comparators that correspond to a single level in $\mathcal{C}$, apply the expander-like construction of the so-called *majority preserver* designed in [4] to each of the blocks. For any fixed constant $r < 1$, by carefully choosing the parameters involved in the majority preservers, we can use the argument of [4] to show that with probability at least $1 - \frac{1}{2n^2}$, for all i, more than rs of the items contained in R_i are the same as that contained in r_i in the fault-free case. The details of the construction and its proof of correctness can be found in [4]. We complete the $O(\log n)$-input $(\rho, \frac{1}{n^2})$-reversal-fault-tolerant sorting network by applying (in parallel) the s-input $(\rho, \frac{1}{2n^2})$-reversal-fault-tolerant r-MAJORITY network of Lemma 3.2.6 to each of the $O(\log n)$ blocks of the form R_i. ∎

3.3 An Optimal EREW Fault-Tolerant Sorting Algorithm

In this section, we present our fault-tolerant sorting algorithm for the EREW PRAM model. In the PRAM model, as pointed out in Section 1.1, we will assume that faults only occur as incorrect answers to comparison queries and that no item is lost in any comparison. In all of the upper bound results in this section, we assume that the fault probability of each comparison is upper bounded by a constant strictly less than $\frac{1}{2}$. Note that when the fault probability is equal to $\frac{1}{2}$, we cannot obtain any useful information from a comparison.

The main result in this section is a fault-tolerant EREW PRAM algorithm for sorting that runs in the optimal $\Theta(\log n)$ time on n processors. This answers an open question posed by Feige, Peleg, Raghavan, and Upfal [7]. The only previously known $o(\log^2 n)$ time fault-tolerant PRAM sorting algorithm on n processors is a randomized algorithm [7].

Theorem 3.3.1 *There exists an explicit deterministic fault-tolerant EREW PRAM sorting algorithm with $\Theta(\log n)$ running time on n processors.*

The following lemma is due to Feige, Peleg, Raghavan, and Upfal [7].

Lemma 3.3.1 ([7]) *There exists an explicit deterministic $(\rho, \rho^{\Theta(\log m)})$-fault-tolerant EREW PRAM algorithm that selects the maximum of m items in $\Theta(\log m)$ time on m processors.*

Proof: See Theorem 20 of [7]. ■

Proof of Theorem 3.3.1: We use the approach of Theorem 3.1.1, with modifications to achieve the claimed $\Theta(\log n)$ running time (recall that the depth bound in Theorem 3.1.1 is $O(\log n \log\log n)$).

For any constant ϵ, by simple majority vote, we easily have a comparison scheme of $O(1)$ running time that yields the correct answer with probability at least $1 - \epsilon$.

In our algorithm, we first use the PRAM to simulate the partial 1-AKS circuit. In order to apply Corollary 2.3.1, we need to make sure that the fault probability of each comparison is upper bounded by a sufficiently small constant. This can be achieved by the majority-vote scheme, which only causes a constant slowdown. By Corollary 2.3.1, all of the outputs from the PRAM simulation of the partial 1-AKS circuit can be partitioned as $X = S \bigcup X_1 \bigcup \ldots \bigcup X_{\bar{n}}$ such that all the items in X_i are smaller than all the items in X_j for $i < j$, where $\bar{n} = \Theta(n^{1-\beta})$.

In the proof of Theorem 3.1.1, we went on to use the approximate-insertion circuit of Lemma 3.1.1. Here, in order to achieve the claimed running time, we need a better insertion scheme so that we will not have to deal with the boundary problem that occurred in the proof of Theorem 3.1.1. In particular, we use the following non-oblivious strategy. In parallel, we apply the selection algorithm of Lemma 3.3.1 to X_i for each $i \leq \bar{n}$. Let M_i be the maximum item in X_i as reported by the selection algorithm. We use all n processors to sort the set

$$P = S \bigcup \{M_1, M_2, \ldots, M_{\bar{n}}\}.$$

Since we have n processors and P contains at most $\bar{n} + |S| = O(n^{\frac{3}{4}})$ items, we can sort P with probability at least $1 - \frac{1}{n^2}$ by simulating the m-input, $O(\log m)$-time, and $O(m \log m)$-register $(\rho, \rho^{\Theta(\log m)})$-destructive-fault-tolerant sorting network designed in [4] with $m = \bar{n} + |S|$.

Based on the sorted order of P, we can derive the approximately correct position for each item in S. In particular, we can partition X as

$$X = \bigcup_{1 \leq i \leq \bar{n}} Y_i$$

where $Y_i = X_i \bigcup \{s \in S \mid M_{i-1} < s \leq M_i\}$ for $i < \bar{n}$, and $Y_{\bar{n}} = X_{\bar{n}} \bigcup \{s \in S \mid s > M_{\bar{n}-1}\}$ (M_0 is assumed to be $-\infty$). It is easy to see that

$$\Theta(n^{\beta}) = |X_1| \leq |Y_i| \leq |S| + |X_1| \leq O(n^{\frac{3}{4}}) + \Theta(n^{\beta}) = \Theta(n^{\beta}), \tag{3.24}$$

where β is the constant specified in Corollary 2.3.1. For $i < j$, all items in Y_i are smaller than all items in Y_j provided that the simulation of the partial 1-AKS circuit and the sorting of P are both done correctly. Note that this output order is "cleaner" than that obtained after the first round of the circuit of Theorem 3.1.1: in the present case, we have no boundary problems to cope with.

To sort the items within Y_i recursively, we have used in Theorem 3.1.1 a more and more intensive application of the replication technique for smaller and smaller set of items in order to guarantee that each of the $O(\log\log n)$ rounds works with probability at least $1 - \frac{1}{n^2}$, instead of $1 - \frac{1}{O(n^{\alpha'})}$. To achieve the $O(\log n)$ running time, here we use an adaptive approach to avoid the $O(\log\log n)$ running time blowup caused by this replication technique.

In parallel, we apply a $(\rho, \frac{1}{|Y_i|})$-fault-tolerant EREW sorting algorithm to Y_i for each $i \leq \bar{n}$. A standard Chernoff bound argument shows that the number of unsorted groups of the form Y_i is at most $\bar{n}\frac{1}{|Y_i|} + \bar{n}^{\frac{3}{4}}$ with probability at least $1 - e^{-\frac{\bar{n}^{\frac{2}{4}}}{2}}$. By inequality 3.24, this means that with probability at most $1 - \frac{1}{n^2}$, the number of unsorted groups of the form Y_i is at most $O(\bar{n}^{\frac{3}{4}})$.

We now detect which groups of the form Y_i remain unsorted. For each i in parallel, we first assume that the order for Y_i reported by the recursive sorting algorithm is correct. Then, we check the correctness of the order by repeatedly comparing adjacent items $O(\log n)$ times. With probability at least $1-\frac{1}{n^2}$, we will detect all of the $O(\bar{n}^{\frac{3}{4}})$ unsorted groups. The total number of items contained in such unsorted groups is at most $O(\bar{n}^{\frac{3}{4}}n^{\beta}) = O(n^{\frac{3}{4}(1-\beta)+\beta}) = O(n^{\frac{3+\beta}{4}})$. Since we have n processors to sort the $O(n^{\frac{3+\beta}{4}})$ unsorted numbers, we can proceed by simulating the m-input, $O(\log m)$-time, and $O(m \log m)$-register $(\rho, \rho^{\Theta(\log m)})$-destructive-fault-tolerant sorting network designed in [4] with $m = O(n^{\frac{3+\beta}{4}})$, which succeeds with probability at least $1 - \frac{1}{n^2}$,

It is easy to see that the failure probability of the whole algorithm is $O(\frac{1}{n^2}) \leq \frac{1}{n}$. Furthermore, this construction leads to the recurrence

$$T(n) = O(\log n) + T\left(O(n^{\beta})\right),$$

where $T(m)$ denotes the time of the optimal $(\rho, \frac{1}{m})$-fault-tolerant EREW sorting algorithm on m processors. Solving this recurrence, we find that $T(n) = O(\log n)$. ■

3.4 Worst-Case Fault-Tolerance

In this section, we extend our results for random faults to construct worst-case fault-tolerant sorting circuits, networks, and algorithms. All previous work on worst-case faults seems to have focused on passive faults [23, 24, 26]. We are not aware of any previous work on sorting networks that are tolerant to worst-case reversal or destructive faults or PRAM sorting algorithms that are tolerant to worst-case faults. We define a circuit, network, or algorithm to be *k-fault-tolerant* if the circuit, network, or algorithm remains a sorting circuit, network, or algorithm even if any k or fewer comparators (or comparisons in the case of an algorithm) are faulty.

Throughout this section, we will use the following simple scheme to construct k-fault-tolerant circuits, networks, and algorithms: Take a (ρ, ϵ)-fault-tolerant circuit, network, or algorithm where $\epsilon = \rho^{k+1}$. Despite different technical details for different fault and computation models, our hope is that such a circuit, network, or algorithm will be able to tolerate up to k worst-case faults. This is formally proved in the next lemma, where Q should be interpreted as a sorting related problem.

Lemma 3.4.1 *Let $\mathcal{A}$ be a circuit, network, or algorithm for solving a certain problem Q. If $\mathcal{A}$ is (ρ, ϵ)-fault-tolerant, then $\mathcal{A}$ is k-fault-tolerant for $k = \log_\rho \epsilon - 1$.*

Proof: Assume for the purposes of contradiction that $\mathcal{A}$ is not k-fault-tolerant. Then, there exists a set S of k or fewer comparators (or comparisons if $\mathcal{A}$ is an algorithm) such that if all the comparators (or comparisons) in S are faulty and all of the other comparators (or comparisons) not in S are correct then the resulting faulty version of $\mathcal{A}$ fails to solve the problem Q. On the other hand, if we set each comparator (or comparison) in S be faulty with probability ρ and each comparator (or comparison) not in S be faulty with probability 0, then $\mathcal{A}$ fails to solve Q with probability $\rho^{|S|} = \rho^k = \rho^{\log_\rho \epsilon - 1} > \epsilon$. This contradicts the assumption that $\mathcal{A}$ is

(ρ, ϵ)-fault-tolerant. ■

The remainder of this section is organized into four subsections. The first three subsections contain results for passive faults, reversal faults, and EREW PRAM algorithms, respectively. We conclude in the last subsection by pointing out that all of the results in this section can be proved independent of the fault-tolerance of the l-AKS circuit that is proved in Theorem 2.3.1.

3.4.1 Results for Passive Faults

In this subsection, we construct a k-passive-fault-tolerant sorting circuit with nontrivial depth. Although the tight bound on the size of worst-case passive-fault-tolerant sorting circuits was derived by Yao and Yao [26] in 1985, our result provides the first asymptotically nontrivial upper bound on the depth, and is itself asymptotically optimal over a large range of k.

Theorem 3.4.1 *There exists an explicit construction of a k-passive-fault-tolerant sorting circuit with $O(\log n + k \log \frac{\log n}{\log k})$ depth.*

There is a trivial lower bound of $\Omega(\log n + k)$ on the depth of k-passive-fault-tolerant sorting circuits. The $\Omega(\log n)$ lower bound follows from the trivial $\Omega(\log n)$ lower bound for the fault-free case. In order for a circuit to tolerate k passive-faults, each register r in the circuit must be connected to at least $k+1$ comparators. (Otherwise, all of the comparators associated with r could be faulty, and the circuit would fail if the item input to r should not be output to r.) This implies an $\Omega(k)$ lower bound on the depth. Combining the $\Omega(k)$ and $\Omega(\log n)$ lower bounds, we have proved the $\Omega(\log n + k)$ lower bound on the depth. Therefore, the upper bound of Theorem 3.4.1 is actually tight when $k = O(\frac{\log n}{\log \log n})$ or $k = \Omega(n^\alpha)$ for any constant α.

In fact, when $k = O(\log n)$, by taking the circuit of Theorem 3.1.1 and applying Lemma 3.4.1 with ρ set to a constant, we immediately get a k-passive-fault-tolerant sorting circuit with $O((\log n + k) \log \log n)$ depth. For larger k, we can construct a k-passive-fault-tolerant sorting circuit with $O((\log n + k) \log \log n)$ depth by replicating each comparator of the $O(\log n)$-passive-fault-tolerant sorting circuit $\Omega(\frac{k}{\log n})$ times. However, to achieve the better depth bound claimed in Theorem 3.4.1, we will use a slightly different approach.

Lemma 3.4.2 *For some constant $\alpha < 1$, there exists an explicit construction of a k-passive-fault-tolerant m-input m^α-approximate-sorting circuit with depth $O(\log m + k)$.*

Proof: An m-input $(\log m)$-passive-fault-tolerant m^α-approximate-sorting circuit can be easily constructed by applying Lemmas 3.4.1 and 3.1.2. This proves the theorem for the case $k \le \log m$. When $k > \log m$, the claimed circuit can be constructed by replicating each comparator of the $(\log m)$-passive-fault-tolerant circuit $\left\lceil \frac{k}{\log m} \right\rceil$ times. ■

Lemma 3.4.3 *There exists an explicit construction of an m-input $O(m)$-passive-fault-tolerant sorting circuit with $O(m)$ depth.*

Proof: This follows from Lemmas 2.3.1 and 3.4.1. (Recall that as pointed in the proof of Lemma 2.3.1, for passive faults, the circuit of Lemma 2.3.1 is actually a $(\rho, \rho^{\Theta(\log m)})$-fault-tolerant circuit for sorting, as opposed to approximate-sorting.) ■

Proof of Theorem 3.4.1: We first apply the k-passive-fault-tolerant n^α-approximate-sorting circuit of Lemma 3.4.2 to all n items. The outputs of this circuit form an $O(n^\alpha)$-approximate-sorted list. Then, we apply two sets of $O(n^\alpha)$-input $O(n^{\alpha^2})$-approximate-sorting circuits of Lemma 3.4.2 so that the boundaries of the second set of circuits fall in the centers of the first set of circuits. We can repeat this construction within smaller and smaller groups until the group size is no more than k, where we can finish up by using the circuit of Lemma 3.4.3. Detailed parameter choices are similar to those in the proof of Theorem 3.1.1. This construction leads to the following formula on the depth $D(n)$:

$$D(n) = O(k + \log n) + O(k + \log n^\alpha) + \cdots + O(k + \log n^{\alpha^i}) + O(k)$$

where i is the smallest integer such that $O(n^{\alpha^i}) \leq k$. This yields $D(n) = O(\log n + k \log \frac{\log n}{\log k})$. ■

3.4.2 Results for Reversal Faults

We begin this subsection by proving that $\Delta \geq k$ for any k-reversal-fault-tolerant Δ-approximate-sorting circuit. Then, we prove a lower bound on the depth of any k-reversal-fault-tolerant k-approximate-sorting circuit. We continue with a k-reversal-fault-tolerant k-approximate-sorting circuit with nontrivial depth and size. The most important result in this subsection is the construction of a k-reversal-fault-tolerant k-approximate-sorting network with nontrivial size and depth.

Theorem 3.4.2 *For any $k < n$, there is no k-reversal-fault-tolerant $(k-1)$-approximate-sorting circuit.*

Proof: We focus on an arbitrary n-input circuit $\mathcal{C}$ and show that some faulty version of $\mathcal{C}$ with at most k faults cannot $(k-1)$-approximate-sort. Since the theorem is concerned with the functionality of $\mathcal{C}$, and not the depth of $\mathcal{C}$, we can assume without loss of generality that each level of $\mathcal{C}$ contains only one comparator. We use the notions of fault patterns and register segments as defined on pages 10 and 70, respectively. Also, we use $\mathcal{C}(F)$ to denote the faulty version of $\mathcal{C}$ specified by fault pattern F. From the top to the bottom, we label the n registers of $\mathcal{C}$ by $r_1, r_2, \ldots, r_n$. Let $r_{i,j}$ denote the register segment of r_i between levels j and $j+1$ for $j = 0, 1, \ldots, d$, where d is the depth of $\mathcal{C}$.

In what follows, we assume that x is the item that is supposed to be output to register r_1 when a permutation of $\{1, 2, \ldots, n\}$ is input to $\mathcal{C}$. Let

$$I_k(j) = \left\{ i \;\middle|\; \begin{array}{c} \exists \text{ a fault pattern } F \text{ with } k \text{ or fewer faults and a permutation } \pi \text{ of} \\ \{1, 2, \ldots, n\} \text{ such that } r_{i,j} \text{ contains } x \text{ when } \pi \text{ is input to } \mathcal{C}(F) \end{array} \right\}.$$

We next prove that

$$|I_k(j)| \geq k + 1 \qquad \text{for } k < n. \tag{3.25}$$

The fact that some faulty version of $\mathcal{C}$ does not $(k-1)$-approximate-sort follows from the preceding inequality with $j = d$, since x is supposed to be output to r_1 and there are at most k registers within $k-1$ of r_1.

We prove inequality 3.25 by induction on j. The base case $j = 0$ is trivial, because each of the n input register segments can contain item x.

Assuming that inequality 3.25 holds up to $j-1$, we prove that inequality 3.25 holds for j. Assume

$$I_k(j-1) = \{i_1, i_2, \ldots, i_s\}.$$

By the induction hypothesis, $s \geq k+1$. Assume that the unique comparator at level j connects registers r_u and r_v. Since the comparator between r_u and r_v is the only comparator at level j, we have

$$I_k(j) \supseteq I_k(j-1) - \{u\} \;\text{ or }\; I_k(j) \supseteq I_k(j-1) - \{v\}. \qquad (3.26)$$

There are three cases to consider.

Case 1: $u \notin I_k(j-1)$ and $v \notin I_k(j-1)$. In this case, $I_k(j) \supseteq I_k(j-1)$, and inequality 3.25 follows from the induction hypothesis.

Case 2: Exactly one of u and v is in $I_k(j-1)$. Without loss of generality, we assume that $u \in I_k(j-1)$ and $v \notin I_k(j-1)$. Hence, on some input permutation, we can force $r_{u,j-1}$ to contain x by using up to k faults before and including level $j-1$. This will cause either $r_{u,j}$ or $r_{v,j}$ to contain x. Hence, by relation 3.26, either $I_k(j) \supseteq I_k(j-1)$ or $I_k(j) \supseteq I_k(j-1) \cup \{v\} - \{u\}$. In either case, inequality 3.25 holds for j.

Case 3: $u \in I_k(j-1)$ and $v \in I_k(j-1)$. In this case, $I_k(j-1) \supseteq I_{k-1}(j-1) \cup \{u, v\}$. Hence, if neither u nor v is in $I_{k-1}(j-1)$, then

$$|I_k(j-1)| \geq |I_{k-1}(j-1)| + 2 \geq k+2,$$

where the second inequality follows from the induction hypothesis. This, together with relation 3.26, implies that inequality 3.25 holds for j. Therefore, we only need to check the case where either u or v is in $I_{k-1}(j-1)$. Without loss of generality, we assume that $u \in I_{k-1}(j-1)$. Under this assumption, on some input permutation, we can force $r_{u,j-1}$ to contain x by spending up to $k-1$ faults before and including level $j-1$. Then, by setting the unique comparator at level j to be either faulty or correct, we can force x to enter either $r_{u,j}$ or $r_{v,j}$, whichever we desire. This means $u \in I_k(j)$ and $v \in I_k(j)$. Hence, by relation 3.26, $I_k(j) \supseteq I_k(j-1)$, which together with the induction hypothesis implies the correctness of inequality 3.25 for j. This finishes the inductive proof of inequality 3.25 as well as the proof of Theorem 3.4.2.

■

Theorem 3.4.3 *Any k-reversal-fault-tolerant k-approximate-sorting circuit has $\Omega(k \log \frac{n}{k})$ depth.*

Proof: Let $\mathcal{C}$ be a k-reversal-fault-tolerant k-approximate-sorting circuit with depth d. We will use the notion of register segments as defined on page 70. In particular, let $r(i,l)$ be the register segment of $\mathcal{C}$ that contains i between levels l and $l+1$ when the identity permutation $(1, 2, \cdots, n)$ is input to the non-faulty version of $\mathcal{C}$. For example, $r(i,0)$ is the ith input register segment. For ease of notation, we assume $r(i,l) = r(i,0)$ for $l < 0$.

Claim 3.4.1 *If circuit $\mathcal{C}$ contains at most k faults, then for any integers $l \geq 0$ and $i \in [1,n]$, $r(i, d-lk)$ can only contain an item in $[i - 2^l k, i + 2^l k]$.*

We will prove the claim by induction on l. For the base case where $l = 0$, the claim is true since (the non-faulty) $\mathcal{C}$ should k-approximate-sort all permutations, including the identity permutation. Assuming that the claim holds for $l-1$, we now prove the claim for l.

Recall that as defined on page 75, a register segment y is a descendant of a register segment x if and only if the item contained in x could subsequently enter y in some faulty or non-faulty version of $\mathcal{C}$. Also, following the notations defined on page 75, we use $D(x,j)$ to denote the set of descendants of x between levels j and $j+1$. We next prove that

$$D\left(r(i,d-lk),d-(l-1)k\right) \subseteq \{r(j,d-(l-1)k) \mid j \in [i-2^{l-1}k, i+2^{l-1}k]\}. \quad (3.27)$$

Assume for the purposes of contradiction that the above inequality does not hold. Then, there exists

$$j \notin [i-2^{l-1}k, i+2^{l-1}k] \quad (3.28)$$

such that $r(j,d-(l-1)k)$ is a descendant of $r(i,d-lk)$. Hence, there is a path from $r(i,d-lk)$ to $r(j,d-(l-1)k)$ that spans at most k levels in $\mathcal{C}$. On the other hand, $r(i,d-lk)$ receives i when the identity permutation is input to any version of $\mathcal{C}$ that is fault-free in the first $d-lk$ levels. By forcing each of the k or fewer comparators along the path from $r(i,d-lk)$ to $r(j,d-(l-1)k)$ to be faulty or correct, as appropriate, we can force $r(j,d-(l-1)k)$ to receive i. However, by the induction hypothesis, $r(j,d-(l-1)k)$ can only receive an item in $[j-2^{l-1}k, j+2^{l-1}k]$ when $\mathcal{C}$ contains at most k faults. Hence, we have $i \in [j-2^{l-1}k, j+2^{l-1}k]$, which contradicts relation 3.28. This proves relation 3.27.

By relation 3.27, if, on any input permutation, $r(i,d-lk)$ receives an item j' when $\mathcal{C}$ contains k or fewer faults, then j' is eventually moved from $r(i,d-lk)$ to a register segment $r(j,d-(l-1)k)$ for some $j \in [i-2^{l-1}k, i+2^{l-1}k]$. On the other hand, by the induction hypothesis, $r(j,d-(l-1)k)$ can only receive items in $[j-2^{l-1}k, j+2^{l-1}k]$ when $\mathcal{C}$ contains at most k faults. Thus, $j' \in [j-2^{l-1}k, j+2^{l-1}k] \subseteq [i-2^{l}k, i+2^{l}k]$. This finishes the inductive step for proving Claim 3.4.1.

By Claim 3.4.1 with $l = \left\lceil \frac{d}{k} \right\rceil$, we find that the input register segment $r(1,0)$ can only contain numbers in $[1-2^{l}k, 1+2^{l}k]$. On the other hand, the item input to $r(1,0)$ can be any number in $[1,n]$. Hence, we have $1+2^{l}k \geq n$, which yields $l = \left\lceil \frac{d}{k} \right\rceil \geq \log \frac{n-1}{k}$. This completes the proof of Theorem 3.4.3. ■

We remark that the above proof can be extended to obtain the same lower bound for the much simpler problem of k-approximate-insertion. This requires slightly more efforts, and we will not prove this fact in the thesis.

Theorem 3.4.4 *There exists an explicit construction of a k-reversal-fault-tolerant k-approximate-sorting circuit with $O(k \log n + k \log^2 k)$ depth and $O(n(\log n + k \log \frac{\log n}{\log k} + k \log^2 k))$ size.*

Lemma 3.4.4 *There exists an explicit construction of an m-input k-reversal-fault-tolerant k-approximate-insertion circuit with $O(k \log m)$ depth and $O(mk)$ size.*

Proof: By using the proof technique of Lemma 2.3.1, it is easy to prove that for any constant $c > 1$, there exists a ck-input odd-even transposition circuit of $O(k)$ depth that is a k-reversal-fault-tolerant k-approximate-sorting circuit. Using this fact and the argument for Lemma 3.2.3, it is easy to show that the circuit of Lemma 3.2.3 with $l = O(k)$ has the desired property. Note that the assumption $l \geq \log \frac{m}{l}$ in Lemma 3.2.3 was used only in the probabilistic analysis. Hence, the assumption is not necessary when the circuit is subject to worst-case faults. In fact, for worst-case faults, all of the odd-even transposition circuits involved in the analysis are k-approximate-sorting circuits. ■

Lemma 3.4.5 *For $k \leq m^{\frac{1-\beta}{2}}$ (where β is the constant specified in Corollary 2.3.1), there exists an explicit construction of an m-input k-reversal-fault-tolerant m^{α}-approximate-sorting circuit with $O(k \log m)$ depth and $O(m(k + \log m))$ size where $\alpha < 1$ is a constant.*

Proof: By Lemma 3.4.1 and Corollary 2.3.1, for some $l = O(\max\{\frac{k}{\log m}, 1\})$, an m-input partial l-AKS circuit will move every item, except those in a set S with $O(m^{\frac{3}{4}})$ items, to within $O(m^{\beta})$ of the correct position, even in the presence of up to k worst-case reversal faults. This part has $O(\log m + k)$ depth and $O(m(\log m + k))$ size. We complete the claimed circuit by applying a set of k-approximate-insertion circuits of Lemma 3.4.4 in a fashion similar to that of Lemma 3.1.2. This part has $O(k \log m)$ depth and $O(mk)$ size. ■

Lemma 3.4.6 *There exists an explicit construction of an m-input k-reversal-fault-tolerant k-approximate-sorting circuit with $O(k \log^2 m)$ depth and $O(mk \log^2 m)$ size.*

Proof: By using the argument for Lemma 3.2.5, it is not hard to show that the circuit of Lemma 3.2.5 with $l = k$ is a k-reversal-fault-tolerant $O(k)$-approximate-sorting circuit with $O(k \log^2 m)$ depth and $O(mk \log^2 m)$ size. (The assumption $l \geq c \log m$ was used in the probablistic argument for Lemma 3.2.5; such an assumption is not needed for worst-case faults since we can actually argue that all of the constituent odd-even transposition circuits are k-reversal-fault-tolerant.) Given the $O(k)$-approximate-sorted list thus produced, we can finish up by applying an odd-even transposition circuit of $O(k)$ depth to the entire list. By the argument of Lemma 2.3.1, we can show that after $O(k)$ steps of the odd-even transposition circuit, every item is within k of the correct position. ■

Proof of Theorem 3.4.4: Given the circuit of Lemma 3.4.5, we can construct the claimed circuit in a manner similar to that used in the proof of Theorem 3.2.3. In particular, we repeatedly apply the circuit of Lemma 3.4.5 until $m = k^{\frac{2}{1-\beta}}$, at which point we can directly apply the circuit of Lemma 3.4.6. The depth of the entire circuit is at most

$$\begin{aligned} & O(k \log n) + O(k \log n^{\alpha}) + \cdots + O(k \log n^{\alpha^i}) + \cdots + O(k \log k^{\frac{2}{1-\beta}}) + O(k \log^2 k) \\ = \; & O(k \log n + k \log^2 k). \end{aligned}$$

The size of the entire circuit is at most

$$\begin{aligned} & O(n(k + \log n)) + O(n(k + \log n^{\alpha})) + \cdots + O(n(k + \log n^{\alpha^i})) + \cdots + \\ & + O(n(k + \log k^{\frac{2}{1-\beta}})) + O(nk \log^2 k) \\ = \; & O(n(\log n + k \log \frac{\log n}{\log k} + k \log^2 k)). \end{aligned}$$

■

Theorem 3.4.5 *There exists an explicit construction of a k-reversal-fault-tolerant sorting network with $O(n(\log n + k \log \frac{\log n}{\log k} + k^{\log_2 3}))$ size.*

The definition of the r-MAJORITY function in the next lemma can be found on page 85.

Lemma 3.4.7 ([9]) *There exists an explicit construction of a k-reversal-fault-tolerant network with $O(k^{\log_2 3})$ size that computes the r-MAJORITY function with $O(k)$ inputs for some constant $r \in (\frac{1}{2}, 1)$.*

Proof: See [9]. ■

Proof of Theorem 3.4.5: We first apply the k-approximate-sorting circuit in Theorem 3.4.4. Then, we use the Assaf-Upfal [4] technique followed by n r-MAJORITY networks with $O(k)$ inputs each, as described in Lemma 3.4.7, for some constant $r \in (\frac{1}{2}, 1)$. The details are similar to those in the proof of Theorem 3.2.4. ■

3.4.3 An Optimal EREW k-Fault-Tolerant Sorting Algorithm

This section contains an optimal k-fault-tolerant EREW PRAM sorting algorithm.

Theorem 3.4.6 *There exists an explicit EREW PRAM k-fault-tolerant sorting algorithm that runs in asymptotically optimal $\Theta(\log n + k)$ time on n processors.*

Proof: We first prove that any k-fault-tolerant PRAM sorting algorithm on n processors runs in $\Omega(\log n + k)$ time. Since any sorting algorithm uses $\Omega(n \log n)$ comparisons, it is sufficient to prove a lower bound of $\Omega(k)$. In the PRAM model, each processor can only make c comparisons at each single step, where c is a constant. Assume for the purposes of contradiction that a k-fault-tolerant sorting algorithm runs in at most $\frac{k}{c}$ steps. Then there exists an output item x that is involved in k or fewer comparisons. If these k or fewer comparisons for x are all faulty, then we actually know nothing about the rank of x. This is a contradiction.

According to Lemma 3.4.1, a $(\log n)$-fault-tolerant EREW sorting algorithm $\mathcal{A}$ with $O(\log n)$ running time on n processors can be easily constructed from the algorithm of Theorem 3.3.1. For $k < \log n$, algorithm $\mathcal{A}$ satisfies all the claimed properties. For $k > \log n$, we construct another algorithm $\mathcal{A}'$ with the claimed property by simulating $\mathcal{A}$, as follows. In $\mathcal{A}'$, we replace each comparison query of $\mathcal{A}$ by $\frac{2k+1}{\log n}$ comparisons, and we use the majority of the answers as the final answer to that query. In order to make an answer to a comparison query in $\mathcal{A}$ be incorrect, an adversary would need to spend more than $\frac{k}{\log n}$ faults for $\mathcal{A}'$. With up to k faults for $\mathcal{A}'$, the adversary can force at most $\frac{k}{\frac{k}{\log n}} = \log n$ incorrect comparison answers in $\mathcal{A}$. Since $\mathcal{A}$ can tolerate up to $\log n$ faults, $\mathcal{A}'$ works correctly with up to k faults. Since we simulate each comparison of $\mathcal{A}$ by $\frac{2k+1}{\log n}$ comparisons in $\mathcal{A}'$, the running time of $\mathcal{A}'$ is $\Theta(\frac{k}{\log n})$ times the running time of $\mathcal{A}$. Hence, algorithm $\mathcal{A}'$ runs in $\Theta(k)$ time. ∎

3.4.4 Remarks

Thus far, all of our worst-case fault-tolerant constructions of circuits, networks, and algorithms have been based on the corresponding constructions for random faults. Since our results for random faults are dependent on the fault-tolerance properties of the l-AKS circuit proved in Theorem 2.3.1, our results for worst-case faults are also dependent on Theorem 2.3.1.

Given the difficulty of proving Theorem 2.3.1, it is worth pointing out that all of our results for worst-case faults can be proved independent of Theorem 2.3.1. For this purpose, the replacement of Theorem 2.3.1 is a lemma that is essentially due to Yao and Yao [26], who proved the lemma for passive faults. In what follows, we define the Hamming distance $D(\mathbf{x}, \mathbf{y})$ of any two 0-1 sequences $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ to be the number of positions where $\mathbf{x}$ and $\mathbf{y}$ differ. For any (possibly faulty) circuit $\mathcal{C}$, we use the notation $\mathcal{C}(\mathbf{x})$ to denote the 0-1 sequence that is output by $\mathcal{C}$ on input $\mathbf{x}$. Yao and Yao's lemma states that for any circuits $\mathcal{C}$ and $\mathcal{C}'$ where $\mathcal{C}'$ is a faulty version of $\mathcal{C}$ with at most k passive or reversal faults, $D(\mathcal{C}(\mathbf{x}), \mathcal{C}'(\mathbf{x})) \leq 2k$ where $\mathbf{x}$ is any 0-1 sequence in $\{0,1\}^n$. Even though Yao and Yao considered passive faults only, their proof can actually be extended to deal with reversal faults. Given Yao and Yao's lemma, we can construct circuits, networks, and algorithms by an approach used in [15]. In particular, we will need circuits that isolate extreme items into a small group of registers. We can construct such circuits by adapting the fault-tolerant minimum-finding algorithm of [7] (see Lemma 3.3.1) or by adapting the passive-fault-tolerant minimum-finding circuit of [6]. The details are not simple and are omitted in this thesis.

Chapter 4

Tight Lower Bounds on the Size of Destructive-Fault-Tolerant Sorting Networks

In this chapter, we prove a tight lower bound of $\Omega(n \log^2 n)$ on the size of destructive-fault-tolerant sorting networks. Somewhat surprisingly, we can show the same lower bound even for the much simplier problem of merging. We also consider worst-case destructive faults, and we prove a lower bound of $\frac{(k+1)n \log n}{4}$ on the size of k-destructive-fault-tolerant sorting or merging networks. This lower bound is tight for $k = O(\log n)$. We also study the trade-off between the depth and width of destructive-fault-tolerant sorting or merging networks. (The *width* of a network is defined to be the number of registers in the network.) These results completely settle the open questions posed by Assaf and Upfal [4] on the size and width of destructive-fault-tolerant sorting networks.

The remainder of the chapter is organized into sections as follows. We start in Section 4.1 by proving the $\Omega(kn \log n)$ lower bound on the size of k-destructive-fault-tolerant sorting or merging networks. We then extend this result in Section 4.2 to prove the $\Omega(n \log^2 n)$ lower bound on the size of destructive-fault-tolerant sorting or merging networks. The material in Section 4.2 represents the most difficult and important result of the chapter. In Section 4.3, we study the trade-off between the depth and width of destructive-fault-tolerant sorting and merging networks.

4.1 Lower Bounds for Worst-Case Faults

In this section, we prove a lower bound on the size of merging and sorting networks that are tolerant to worst-case faults.

Theorem 4.1.1 *The size of any k-destructive-fault-tolerant merging (or sorting) network is at least $\frac{(k+1)n\log n}{4}$.*

We will prove Theorem 4.1.1 by showing a lower bound for merging networks. Without loss of generality, we will assume that n is an exact power of two. In what follows, we will use $\mathcal{M}$ to denote a k-destructive-fault-tolerant merging network that takes two sorted lists $X = (x_1 \leq x_2 \leq \cdots \leq x_{n/2})$ and $Y = (y_1 \leq y_2 \leq \cdots \leq y_{n/2})$ as inputs and that outputs the merged list. Without loss of generality, this means that $\mathcal{M}$ sorts lists of the form $(x_1, y_1, x_2, y_2, \ldots, x_{n/2}, y_{n/2})$ where $x_1 \leq x_2 \leq \cdots \leq x_{n/2}$ and $y_1 \leq y_2 \leq \cdots \leq y_{n/2}$.

To show that $\mathcal{M}$ must have $\frac{(k+1)n\log n}{4}$ comparators, we need the following definitions and lemmas, some of which are extensions of those used by Floyd to prove that a fault-free merging network needs $\Omega(n \log n)$ comparators (see Theorem F on pages 230-232 of [10]).

Given a merging network $\mathcal{M}$ with fault pattern F and a list of integer inputs $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$ to $\mathcal{M}$, we denote the *content* of a register r immediately after level t by $\mathcal{C}(r,t)$.[1] For example, in the network of Figure 1-2, if we label all of the registers from the top to the bottom by r_1, r_2, r_3, and r_4, then $\mathcal{C}(r_3, 1) = 0$ and $\mathcal{C}(r_3, 2) = \mathcal{C}(r_3, 3) = 1$. We define the *history* of $\mathcal{M}$ given F and Π to be the collection of register contents $\mathcal{C}(r,t)$ taken over all registers and all levels. The following lemma

[1]The concept of a content here is essentially the same as that defined on page 22, but our notation for a content in this chapter is different from that used in Chapter 2. A formal definition of a fault pattern can be found on page 10.

shows how the history of a network computation can be influenced by a fault at a single output of a comparator.

Lemma 4.1.1 *Given any $\mathcal{M}$, Π, F, and F', let $\mathcal{C}(r,t)$ denote the content of register r immediately after level t for $\mathcal{M}$ given F and Π, and let $\mathcal{C}'(r,t)$ denote the content of register r immediately after level t for $\mathcal{M}$ given F' and Π. If F' is identical to F except that one comparator C on level l with output registers p and q is modified in F' so that $\mathcal{C}'(q,l) = \mathcal{C}(q,l)$ and $\mathcal{C}'(p,l) \neq \mathcal{C}(p,l)$, then for all r and t*

(1) if $\mathcal{C}(r,t) < \mathcal{C}(p,l)$, then $\mathcal{C}'(r,t) \leq \mathcal{C}(r,t)$,

(2) if $\mathcal{C}(r,t) > \mathcal{C}(p,l)$, then $\mathcal{C}'(r,t) \geq \mathcal{C}(r,t)$.

Proof: For simplicity, let $s = \mathcal{C}(p,l)$ and $s' = \mathcal{C}'(p,l)$. We will only prove the lemma for the case in which $s' < s$. The case in which $s' > s$ can be proved similarly.

We will think of F' as the result of modifying C in F and prove some properties must hold when such modification is made. The network can be divided into two parts: the part before (and including) level l and the part after level l. The former part clearly remains unchanged after C is modified, and the latter part is changed as if one input item to that part were modified. Hence, it suffices to show that the following properties (a) and (b) hold when some input item is changed from s to some $s' < s$ but no comparator is modified.

(a) The content of each register segment[2] whose content was less than s before the modification is not increased.

(b) The content of each register segment whose content was greater than s before the modification remains unchanged.

In a given history, if an input item is changed from s to $s-1$ (but no comparator is modified), then all the input items that are neither s nor $s-1$ move in the network as they did before. Hence, all the register segments containing neither s nor $s-1$ before the modification must contain the same value as before. All the register segments containing $s-1$ before the modification must contain $s-1$, while some register segments containing s before the modification may contain $s-1$ now. This means

[2]The definition of a register segment is given on page 70.

that properties (a) and (b) hold when an input s is changed to $s-1$. If the new input item $s-1$ (the one that was s before) is further changed to $s-2$, then the only new change in the history is that some register segments containing $s-1$ before the modification may contain $s-2$ now. Overall, an input item has been changed from s to $s-2$ and properties (a) and (b) still hold. Since both s and s' are integers, this process can be continued until an input has been changed from s to s'. During the whole process, properties (a) and (b) are never violated. ■

Consider the history of $\mathcal{M}$ on a particular 0-1 input sequence π. A *crossing comparator* of $\mathcal{M}$ with respect to π is defined to be a comparator whose input contents are $\{0,1\}$ (i.e., they are not both 0 or both 1).[3] $\mathcal{M}_0(\pi)$, a subnetwork of $\mathcal{M}$ with respect to π, is constructed as follows. (The subscript 0 is used to denote that it is the part of $\mathcal{M}$ that contains 0 on π.) Take all the register segments and replicators that contain 0 and all the comparators with both inputs containing 0. Replace each crossing comparator of $\mathcal{M}_0$ with respect to π by connecting directly its (unique) input containing 0 and its (unique) output containing 0. $\mathcal{M}_1(\pi)$ can be constructed in a similar fashion. For example, when $\mathcal{M}$ and π are as shown in Figure 1-2, $\mathcal{M}_0(\pi)$ and $\mathcal{M}_1(\pi)$ are illustrated in Figure 4-1.

Unless specified otherwise, we will be particularly interested in the history of $\mathcal{M}$ (when there are *no* faults) on the input sequence

$$(\underbrace{0,0,\ldots,0}_{n/2},\underbrace{1,1,\ldots,1}_{n/2}). \tag{4.1}$$

Therefore, when we talk about crossing comparators, $\mathcal{M}_0$, and $\mathcal{M}_1$ without specifying π, π should be interpreted as the 0-1 permutation given in expression 4.1. In particular, to construct $\mathcal{M}_0$ and $\mathcal{M}_1$, we input the smallest $\frac{n}{2}$ items to the top half of $\mathcal{M}$ and the largest $\frac{n}{2}$ items to the bottom half of $\mathcal{M}$. Also, the smallest $\frac{n}{2}$ items should be contained in $\mathcal{M}_0$, and the largest $\frac{n}{2}$ items should be contained in $\mathcal{M}_1$.

[3]The concept of a crossing comparator defined here is similar to that defined on page 72. But they are not identical in that they are defined with respect to different input sequences. We will slightly change the notion of a crossing comparator again in the next section.

The motivation for defining crossing comparators, $\mathcal{M}_0$, and $\mathcal{M}_1$ can be found in the following lemmas.

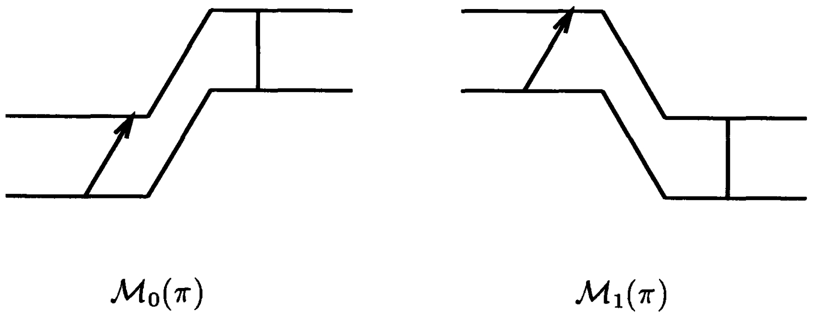

Figure 4-1: The decomposition of the network in Figure 1-2, with respect to the input sequence therein, into $\mathcal{M}_0(\pi)$ and $\mathcal{M}_1(\pi)$. The sloping lines represent direct connections that replace crossing comparators.

Lemma 4.1.2 *If $\mathcal{M}$ is an n-input k-destructive-fault-tolerant merging network, then both $\mathcal{M}_0$ and $\mathcal{M}_1$ are $\frac{n}{2}$-input k-destructive-fault-tolerant merging networks.*

Proof: If we input to $\mathcal{M}$ any sequence

$$(x_1, y_1, x_2, y_2, \ldots, x_{n/4}, y_{n/4}, \underbrace{+\infty, \ldots, +\infty}_{n/2})$$

such that $x_1 \leq x_2 \leq \cdots \leq x_{n/4}$ and $y_1 \leq y_2 \leq \cdots \leq y_{n/4}$ and if none of the crossing comparators of $\mathcal{M}$ are faulty, then, by the definition of $\mathcal{M}_0$, the x's and y's should move within $\mathcal{M}_0$ only, and $\mathcal{M}_1$ has no impact on them. Hence, $\mathcal{M}_0$ must be a k-destructive-fault-tolerant merging network. (Note that an adversary can put no faults at crossing comparators and put up to k faults into $\mathcal{M}_0$.) By a similar argument, we can show $\mathcal{M}_1$ is also a k-destructive-fault-tolerant merging network. ■

Lemma 4.1.3 *If $\mathcal{M}$ is a k-destructive-fault-tolerant merging network, then $\mathcal{M}$ must have at least $\frac{(k+1)n}{4}$ crossing comparators.*

Proof: We will focus on the history of $\mathcal{M}$ on input sequence

$$(1, +\infty, 2, +\infty, \ldots, n/2, +\infty).$$

According to the definition of $\mathcal{M}_0$, $1, 2, \ldots, \frac{n}{2}$ should all be output in the outputs of $\mathcal{M}_0$. In particular, $\frac{n}{4}+1, \ldots, \frac{n}{2}$ should be moved from $\mathcal{M}_1$ to $\mathcal{M}_0$. By definition, each crossing comparator has exactly one $\mathcal{M}_0$-input and one $\mathcal{M}_1$-input. (If an input (output) of a comparator is in $\mathcal{M}_0$, then we call it an $\mathcal{M}_0$-input (output); if an input (output) of a comparator is in $\mathcal{M}_1$, then we call it an $\mathcal{M}_1$-input (output).) Label each crossing comparator by its $\mathcal{M}_1$-input.

Assume for the purposes of contradiction that $\mathcal{M}$ contains less than $\frac{(k+1)n}{4}$ crossing comparators. Then, there exists an integer s, such that $\frac{n}{4}+1 \leq s \leq \frac{n}{2}$ and the total number of crossing comparators labeled by s is at most k.

Let C_1 be a crossing comparator labeled by s that is at the lowest level in $\mathcal{M}$. Since s cannot have moved into $\mathcal{M}_0$ without using a crossing comparator labeled by s, we know that the $\mathcal{M}_0$-input to C_1 does not contain s. Moreover, since C_1 is not faulty in the fault-free network $\mathcal{M}$, one of its outputs contains s and the other output does not contain s. Hence, if we make C_1 be faulty by forcing the $\mathcal{M}_0$-input of C_1 to appear in both outputs of C_1, this will have the effect of replacing the value of s in one of the output registers with a value other than s, which is exactly the scenario described by Lemma 4.1.1. In addition, C_1 can no longer be used to move s from $\mathcal{M}_1$ to $\mathcal{M}_0$.

For any r and t, by Lemma 4.1.1, we know that if $C(r,t) \neq s$ before C_1 is made faulty, then $C(r,t) \neq s$ after C_1 is made faulty. Hence, the number of working crossing comparators labeled by s decreases by at least 1 when C_1 is made faulty.

We next relabel crossing comparators based on the new history when C_1 is faulty and proceed inductively (i.e., we select the next crossing comparator labeled by s, make it faulty, and so forth). The proceeding process terminates when there are no longer any functioning crossing comparators labeled by s. This is accomplished by making at most k crossing comparators faulty. Since there are no longer any crossing comparators that can move s from $\mathcal{M}_1$ to $\mathcal{M}_0$, the network does not successfully complete the merge. Hence, we conclude that $\mathcal{M}$ has at least $\frac{(k+1)n}{4}$ crossing comparators. ∎

Proof of Theorem 4.1.1: For any fixed k, let $S(n)$ denote the size of the smallest n-input k-destructive-fault-tolerant merging network. From the definition of crossing comparators, the construction of $\mathcal{M}_0$ and $\mathcal{M}_1$, and the fact that no crossing comparator in $\mathcal{M}$ appears as a comparator in either $\mathcal{M}_0$ or $\mathcal{M}_1$, we know that

$$\text{size}(\mathcal{M}) \geq \text{size}(\mathcal{M}_0) + \text{size}(\mathcal{M}_1) + |\{\text{crossing comparators in } \mathcal{M}\}|.$$

By Lemmas 4.1.2 and 4.1.3, this means that

$$S(n) \geq 2S\left(\frac{n}{2}\right) + \frac{(k+1)n}{4}$$

for $n \geq 4$. Solving the recurrence, we find that

$$S(n) \geq \frac{n}{2} \cdot S(2) + \frac{(k+1)n\log(n/2)}{4}.$$

Since $S(2) \geq k+1$, this means that $S(n) \geq \frac{(k+1)n\log n}{4}$, as claimed. ■

To conclude this section, we point out that although our lower bound for worst-case destructive faults is tight when $k = O(\log n)$, it can be improved when k is very large, say, exponential in n. we refer the reader to the recent work of Kleitman, Leighton, and Ma [9] for the recent work in this direction.

4.2 Lower Bounds for Random Faults

In this section, we prove a tight lower bound on the size of destructive-fault-tolerant merging and sorting networks. As pointed out in Section 1.3, to get the strongest possible result, p should be interpreted as the failure probability of each comparator in the networks, as opposed to an upper bound on the failure probability. In fact, if p were interpreted as such an upper bound, an $\Omega(n\log^2 n)$ lower bound on the size of destructive-fault-tolerant sorting or merging networks would directly follow from Theorem 4.1.1 with $k = O(\log n)$.

By definition, there exist a fixed number of ways for a comparator to fail in the destructive fault model. When a destructive fault does occur at a comparator, we will assume that each form of the failure appears equally likely. In other words, the probability that a particular form of failure appears at a comparator is $\rho_0 = \frac{\rho}{\alpha}$ where the constant α denotes the total number of possible ways that a comparator can fail. The following theorem is the main result of this chapter.

Theorem 4.2.1 *The size of any (ρ, ϵ)-destructive-fault-tolerant merging (or sorting) network is*

$$\Omega\left(n \log n \frac{\log \frac{1}{\epsilon} + \log \sqrt{n} - \log \log_e \frac{1}{1-\epsilon}}{1 - \log \rho}\right). \tag{4.2}$$

This theorem gives a good lower bound for a large range of ρ and ϵ, and it does not require either ρ or ϵ to be a constant. The most interesting case is that ρ is a non-zero constant and $\epsilon = \frac{1}{\text{poly}(n)}$. In this case, the theorem gives a lower bound of $\Omega(n \log^2 n)$, which is tight [4]. Somewhat surprisingly, however, the theorem gives the same $\Omega(n \log^2 n)$ lower bound even when ϵ is not small. In particular, when ρ is a non-zero constant, the theorem implies the $\Omega(n \log^2 n)$ lower bound even for some extremely small success probability like $1 - \epsilon = e^{-n^{\frac{1}{4}}}$. Hence, even networks that have a tiny chance of surviving the faults must have $\Omega(n \log^2 n)$ size.

In fact, we are going to show a lower bound of the form

$$\Omega\left(n \log n \frac{\log \frac{1}{\epsilon} + \log \sqrt{n} - \log \log_e \frac{1}{1-\epsilon}}{\log \frac{1}{\rho_0}}\right) \tag{4.3}$$

where ρ_0 is the probability that a comparator suffers a particular form of destructive failure. (One can easily check that lower bound 4.3 implies lower bound 4.2. The term "1" in the denominator of expression 4.2 prevents the lower bound from going to $+\infty$ when ρ goes to 1.) We will prove lower bound 4.3 by showing a lower bound of

$$\frac{n \log n \log \epsilon}{4 \log \rho_0}, \tag{4.4}$$

and a lower bound of

$$\frac{n \log n (\log \sqrt{n} - \log \log_e \frac{1}{1-\epsilon})}{4 \log \frac{1}{\rho_0}}. \tag{4.5}$$

When $\epsilon = o\left(\frac{1}{\mathrm{poly(n)}}\right)$, lower bound 4.5 is stronger, and when $\epsilon = \Omega\left(\frac{1}{\mathrm{poly(n)}}\right)$, lower bound 4.4 is stronger.

As a direct application of the proof technique for worst-case faults, if the size of $\mathcal{M}$ does not satisfy lower bound 4.4, then, for any faulty $\mathcal{M}$, we can always produce another faulty network that does not work correctly by forcing at most $\log_{\rho_0} \epsilon = \frac{\log \epsilon}{\log_{\rho_0}}$ comparators to be faulty in a particular way. It is natural to ask if this property is strong enough to show that there is a large fraction of faulty networks that do not work correctly. This can be formulated as the following question. Let S be the set of all sequences with fixed length l, and S_0 be a subset of S. (S will correspond to all fault patterns for a network $\mathcal{M}$, and S_0 will correspond to those fault patterns that keep $\mathcal{M}$ from being a merging network.) If the union of all the hamming balls with origins in S_0 and radius $\log_{\rho_0} \epsilon$ covers S, can we prove $\frac{|S_0|}{|S|} = \Omega(\epsilon)$? (I.e., if every fault pattern is within $\log_{\rho_0} \epsilon$ faults of a bad fault pattern, is the density of bad fault patterns $\Omega(\epsilon)$?) Unfortunately, the answer to this question is in general "No". Hence, to prove our lower bound, we need a better understanding of the structure of the bad fault patterns.

Following the notation of Section 3.2, we use $\mathcal{M}(F)$ to denote the faulty version of $\mathcal{M}$ in which the behavior of each comparator is determined according to fault pattern F. In the proof, we will focus on a particular merging network $\mathcal{M}$ only. Hence, there is a one-to-one correspondence between each fault pattern F and the faulty network $\mathcal{M}(F)$. We call a fault pattern F *good* if $\mathcal{M}(F)$ functions correctly as a merging network, and we call F *bad* otherwise. As in Section 4.1, we want to decompose $\mathcal{M}$ into smaller networks and analyze the behavior of the comparators that connect these small networks. However, unlike in Section 4.1, where we use the fault-free $\mathcal{M}$ to define these concepts, we need to deal with different decompositions and different sets of crossing comparators for different fault patterns. We will use the history of $\mathcal{M}(F)$ on input sequence

$$(\underbrace{0, 0, \ldots, 0}_{n/2}, \underbrace{1, 1, \ldots, 1}_{n/2})$$

to redefine these concepts as follows. A *crossing comparator* of $\mathcal{M}(F)$ is defined to be a comparator whose input contents are $\{0, 1\}$ (i.e., they are not both 0 or both 1).[4] $\mathcal{M}(F)_0$ is constructed as follows. Take all the register segments and replicators that contain 0 and all the comparators with both inputs containing 0. For each crossing comparator with one output containing 0 and the other output containing 1, replace it in $\mathcal{M}(F)_0$ by connecting directly the (unique) input containing 0 to the (unique) output containing 0. For each crossing comparator with both outputs containing 0 (because of a fault), replace it in $\mathcal{M}(F)_0$ by a replicator that copies from the (unique) input register containing 0 to the other (output) register. (We do not include anything in $\mathcal{M}(F)_0$ to represent crossing comparators with both outputs containing 1.) $\mathcal{M}(F)_1$ can be constructed in a similar fashion.

Given $\mathcal{M}(F)_0$, we can construct $\mathcal{M}(F)_{00}$ and $\mathcal{M}(F)_{01}$, and given $\mathcal{M}(F)_1$, we can construct $\mathcal{M}(F)_{10}$ and $\mathcal{M}(F)_{11}$. Working recursively, we can construct $\mathcal{M}(F)_i$ in a similar way for any binary number i with less than $\log n$ bits.

For a fixed $\mathcal{M}$, we can define a partial order on the set of all the comparators in $\mathcal{M}$ by the following rule:

$$C_1 \prec C_2 \quad \text{if and only if} \quad \text{depth}(C_1) < \text{depth}(C_2).$$

We can extend this partial order into a total order. It does not matter how we extend, but we will stick to a fixed extension in the proof.

Proof of Theorem 4.2.1: We start by showing that for any (ρ, ϵ)-destructive-fault-tolerant merging network $\mathcal{M}$,

$$\text{size}(\mathcal{M}) \geq \frac{n \log n \log \epsilon}{4 \log \rho_0}. \tag{4.6}$$

Assume for the purposes of contradiction that inequality 4.6 is not true for $\mathcal{M}$. Then, we will prove that

$$\text{Prob}(F \text{ is bad for } \mathcal{M}) > \epsilon \tag{4.7}$$

[4]This should be compared with the definition of a crossing comparator given on page 105. Also, see the footnote on that page.

where F denotes a randomly generated fault pattern for $\mathcal{M}$. We will prove inequality 4.7 by partitioning the space of all possible fault patterns into small groups and then showing that, within each group, a bad fault pattern will be generated with probability more than ϵ.

Now we will do the most difficult part in the proof, which is to find an appropriate partitioning of the space of the fault patterns. On any fault pattern F, we will use the following 3-step procedure to choose a characteristic set of F, *Char*(F), based on which we are going to define the partition.

Step 1. List all the $\mathcal{M}(F)_i$'s as follows:

$$\mathcal{M}(F), \mathcal{M}(F)_0, \mathcal{M}(F)_1, \mathcal{M}(F)_{00}, \mathcal{M}(F)_{01}, \mathcal{M}(F)_{10}, \mathcal{M}(F)_{11}, \ldots. \tag{4.8}$$

(For any binary numbers i and j with less than $\log n$ bits, we list $\mathcal{M}(F)_i$ before $\mathcal{M}(F)_j$ if i has fewer bits than j, or if i and j have the same number of bits and $i < j$.) Take the first network $\mathcal{M}(F)_i$ in list 4.8 that has less than $\frac{1}{4}n_i\frac{\log \epsilon}{\log \rho_0}$ crossing comparators where $n_i = \frac{n}{2^{l(i)}}$ (where $l(i)$ is the length of i) is the total number of input items to $\mathcal{M}(F)_i$. (Such i does exist since we have assumed that inequality 4.6 is not true.)

By definition, we know that $n_i \geq 2$. In what follows, we will assume $n_i \geq 4$. The case when $n_i = 2$ is easily handled as a special case (or it can be ignored by replacing $\log n$ with $\log n - 1$ in the lower bound).

Step 2. Compute the history of $\mathcal{M}(F)_i$ on input

$$(1, +\infty, 2, +\infty, \ldots, i, +\infty, \ldots, n_i/2, +\infty),$$

and label each crossing comparator by its (unique) $\mathcal{M}(F)_{i1}$-input item. Let $S(j)$ be the set of the crossing comparators labeled by j. Go through the list

$$S\left(\frac{n_i}{4}+1\right), S\left(\frac{n_i}{4}+2\right), \ldots, S\left(\frac{n_i}{2}\right),$$

and choose the first set $S(s)$ such that

$$\sum_{\frac{n_i}{4}+1 \le j \le s} |S(j)| < (s - \frac{n_i}{4}) \frac{\log \epsilon}{\log \rho_0}. \tag{4.9}$$

(Such s does exist, since, by the choice of i, $\mathcal{M}(F)_i$ contains less than $\frac{1}{4} n_i \frac{\log \epsilon}{\log \rho_0}$ crossing comparators, and therefore $s = \frac{n_i}{2}$ must satisfy inequality 4.9.) By the minimality of s, we can conclude the following:

Claim 4.2.1 $|S(s)| < \frac{\log \epsilon}{\log \rho_0}$.

Step 3. We will continue to work on the history of $\mathcal{M}(F)_i$ on input

$$(1, +\infty, 2, +\infty, \ldots, i, +\infty, \ldots, n_i/2, +\infty),$$

and we will choose a characteristic set for F, *Char*(F), from $S(s)$. List all the comparators in $S(s)$ as follows:

$$C_1 \prec C_2 \prec \cdots \prec C_t \tag{4.10}$$

where $\prec$ is the depth-respecting total order described earlier. We first put comparator C_1 into *Char(F)*. Then, we modify the behavior of C_1 (thereby making it faulty in a particular way) so as to make C_1 directly output its $\mathcal{M}(F)_{i0}$-input item to all its $\mathcal{M}(F)_{i0}$-outputs (if any), without changing any $\mathcal{M}(F)_{i1}$-output of C_1.

Before this modification, the $\mathcal{M}(F)_{i1}$-input of C_1 contained s and the $\mathcal{M}(F)_{i0}$-input of C_1 contained non-s, since s could not have moved into $\mathcal{M}(F)_{i0}$ without using a comparator labeled by s. Moreover, if an $\mathcal{M}(F)_{i0}$-output of C_1 did not contain s (which is the $\mathcal{M}(F)_{i1}$-input content of C_1), it had to contain the $\mathcal{M}(F)_{i0}$-input content of C_1. Therefore, this modification has the effect of changing some output content of C_1 from s to non-s. By Lemma 4.1.1, this modification cannot cause any new crossing comparator to be labeled by s. Now C_1 is no longer capable of moving s from $\mathcal{M}(F)_{i1}$ to $\mathcal{M}(F)_{i0}$. Then, we update the history accordingly. In the new history, $C_2, C_3, \ldots, C_t$ are the only remaining comparators that might move s from $\mathcal{M}(F)_{i1}$ to $\mathcal{M}(F)_{i0}$. In this remaining part of list 4.10, we choose the first comparator labeled by s in the new history. We put this comparator into *Char*(F) and modify its behavior as we did for C_1. We then update the history again and continue in this fashion until all comparators in list 4.10 have been dealt with.

This completes the 3-step procedure and the construction of *Char*(F). The final history, in which s is never moved into $\mathcal{M}(F)_{i0}$, corresponds to another fault pattern, and we will call this fault pattern $\tilde{F}$.

Lemma 4.2.1 *For any fault pattern F, the following conditions hold: (1) Char(F) = Char($\tilde{F}$), (2) $\tilde{F}$ is bad, and (3) $|Char(F)| < \frac{\log \epsilon}{\log \rho_0}$.*

Proof: To prove (1), we use the 3-step procedure to determine *Char*($\tilde{F}$) and to show that it is the same as *Char*(F).

We first note that $\mathcal{M}(\tilde{F})$ and $\mathcal{M}(F)$ have the same history on the input list

$$\Pi = (\underbrace{0,\ldots,0}_{n/2},\underbrace{1,\ldots,1}_{n/2}).$$

This is because the changes made in comparators to produce $\tilde{F}$ from F do not affect their performance on Π. Hence, the structure of $\mathcal{M}(\tilde{F})_j$ is the same as the structure of $\mathcal{M}(F)_j$ for all j such that $\mathcal{M}(F)_j$ should be listed before $\mathcal{M}(F)_i$ in list 4.8 (although the functionality of individual comparators may differ). In addition, the crossing comparators for $\mathcal{M}(\tilde{F})_j$ are the same as the crossing comparators for $\mathcal{M}(F)_j$ for all j such that $\mathcal{M}(F)_j$ should be listed before $\mathcal{M}(F)_i$ in list 4.8. Hence, we select the same value of i for $\mathcal{M}(\tilde{F})$ in Step 1 as we do for $\mathcal{M}(F)$.

Next, we show that we pick the same value of s for $\mathcal{M}(\tilde{F})_i$ and $\mathcal{M}(F)_i$ in Step 2. From the construction of $\tilde{F}$, we know that $\tilde{F}$ differs from F (at most) only in comparators that are labeled s in the history of $\mathcal{M}(F)_i$. More precisely, some outputs of these comparators (i.e., the $\mathcal{M}(F)_{i0}$-outputs) that contain s in the history of $\mathcal{M}(F)_i$ may contain non-s values in the history of $\mathcal{M}(\tilde{F})_i$. By Lemma 4.1.1, the effect of these changes is to make (possibly) some values that were s or less in the history of $\mathcal{M}(F)_i$ be smaller in the history of $\mathcal{M}(\tilde{F})_i$, and to make (possibly) some values that were s or greater in the history of $\mathcal{M}(F)_i$ be larger in the history of $\mathcal{M}(\tilde{F})_i$. Hence, the same value of s will be chosen in Step 2 for $\tilde{F}$ as for F. (The reason that we have used the accumulative threshold in Step 2 instead of simply selecting the smallest $S(s)$ should now be apparent.)

According to the description of the 3-step procedure, we can see that the starting history for $\tilde{F}$ at the beginning of the 3-step procedure is the same as the final history at the termination of Step 3 for F. (This can be shown inductively from the lowest level to the highest level). Since the first two steps do not change any comparators, at the beginning of Step 3 for $\tilde{F}$, we have all the comparators in *Char*(F) to start with. As we move along in the history of $\mathcal{M}(\tilde{F})_i$ on input $(1, +\infty, 2, +\infty, \ldots, n_i/2, +\infty)$, we will not make any real change on any comparator in *Char*(F), since, at the termination of the 3-step procedure for F, all the comparators in *Char*(F) have already output their $\mathcal{M}_{i0}$-inputs directly to their $\mathcal{M}_{i0}$-outputs. Furthermore, these comparators are all labeled by s in the history for $\tilde{F}$. Therefore, we have to put all the comparators in *Char*(F) into the characteristic set for $\tilde{F}$. Hence, *Char*(F) = *Char*($\tilde{F}$), as claimed.

To prove (2), we assume for the purposes of contradiction that $\tilde{F}$ is good. Then, the proof technique of Lemma 4.1.2 implies that $\mathcal{M}(\tilde{F})_i$ must function correctly as a merging network. In particular it should work correctly on both input $(\underbrace{0, \ldots, 0}_{n/2}, \underbrace{1, \ldots, 1}_{n/2})$ and input $(1, +\infty, 2, +\infty, \ldots, n_i/2, +\infty)$. Hence, $\mathcal{M}(\tilde{F})_i$ must successfully move s from $\mathcal{M}(\tilde{F})_{i1}$ to $\mathcal{M}(\tilde{F})_{i0}$. However, in the history for $\tilde{F}$, no s can be moved from $\mathcal{M}(\tilde{F})_{i1}$ to $\mathcal{M}(\tilde{F})_{i0}$. This is a contradiction, which means that $\tilde{F}$ is bad.

The correctness of (3) follows from Claim 4.2.1. ■

We are now ready to describe the partition of the space of fault patterns. We group all the fault patterns by the following rule. We put F and F' in the same group if and only if (1) *Char*(F) = *Char*(F'), and (2) the fault patterns F and F' are identical on all the comparators not in *Char*(F).

For any group G, take a fault pattern $F \in G$. By (1) in Lemma 4.2.1 and the construction of $\tilde{F}$, we know that $\tilde{F} \in G$. By (2) in Lemma 4.2.1, we know that $\tilde{F}$ is bad. By (1) and (3) in Lemma 4.2.1, we know that the probability that $\tilde{F}$ occurs is greater than ${\rho_0}^{\frac{\log \epsilon}{\log \rho_0}} = \epsilon$ times the probability that a fault pattern in G occurs. In other words, if a fault pattern in G occurs, there is better than an ϵ chance that it is $\tilde{F}$. Since this is true for all groups, we can thus conclude that the probability that a random fault pattern is bad is greater than ϵ.

This completes the proof of lower bound 4.6. We next show that

$$size(\mathcal{M}) \geq n \log n \frac{\log \sqrt{n} - \log \log_e \frac{1}{1-\epsilon}}{4 \log \frac{1}{\rho_0}}. \qquad (4.11)$$

We will divide the network into blocks of size $\sqrt{n}$ and "pump up" the failure probability by showing that most of the blocks must behave well in order for the overall circuit to work. In particular, we will partition the space of fault patterns into groups and use a conditional probabilistic argument.

For each fault pattern F, we can decompose $\mathcal{M}(F)$ into $\sqrt{n}$ networks, such that each of them has $\sqrt{n}$ inputs and is of the form

$$\mathcal{M}(F)_i$$

where $i \leq \sqrt{n}$ is a binary number with $\frac{\log n}{2}$ bits. By doing so, we have removed many comparators from $\mathcal{M}$. These removed comparators are crossing comparators of many different networks that are larger than the networks with $\sqrt{n}$ inputs that we are currently interested in. We use *Cross*(F) to denote the set of all these removed crossing comparators.[5] We put F and F' in the same group if and only if (1) *Cross*(F) = *Cross*(F'), and (2) F and F' are the same on all the comparators in *Cross*(F).

If, within each group G, the probability that a fault pattern $F \in G$ is good is less than $1-\epsilon$, then a randomly generated fault pattern will be good with probability less than $1-\epsilon$. This is a contradiction to the fact that a randomly generated F is good with probability at least $1-\epsilon$. Therefore, there exists a group G_0 such that

$$\text{Prob}(F \text{ is good} \mid F \in G_0) \geq 1-\epsilon.$$

[5]The notation *Cross*(F) was used to denote a slightly different set of comparators on page 72.

Proof of Theorem 4.3.1: The *Column-Sort* algorithm in [12] arranges all the items in an $r \times s$ matrix where $r \geq 2s^2$, and it consists of 8 phases. Phases 1, 3, 5, 7 sort each column of the matrix and phases 2, 4, 6, 8 permute the matrix in a fixed manner. The only property of *Column-Sort* that we need here is the fact that if we can sort all the columns of the matrix in T steps, then by applying *Column-Sort* we can sort all the items in the matrix in $O(T)$ steps.

Assaf and Upfal [4] have shown how to build a $(\rho, \frac{1}{\text{poly}(n)})$-destructive-fault-tolerant sorting network with width $O(n \log n)$ and depth $O(\log n)$. Hence, for a given w between n and $n \log n$, we can use a network with width w and depth $O(\log(\frac{w}{\log n})) = O(\log n)$ to sort $\frac{w}{\log n}$ items first. (At the same time, we need to keep all other items in some other registers. This can be done as long as we keep enough, say, $2n$, registers.) Then, keep this sorted list of $\frac{w}{\log n}$ items in some registers and work on the next group of $\frac{w}{\log n}$ items, etc. We will have worked on all the groups after $O(\frac{n}{w/\log n}) = O(\frac{n \log n}{w})$ rounds. This finishes the first phase of *Column-Sort* with depth $O(\frac{n \log^2 n}{w})$. To implement the second phase of *Column-Sort*, we can hard-wire in a permutation. Then, we finish the remaining phases in a similar way. The overall depth is $O(\frac{n \log^2 n}{w})$. ■

Theorem 4.3.2 *For any $n \leq w \leq nk$ and $k = O(\log n)$, there exists an explicit construction of a k-destructive-fault-tolerant sorting network with width $O(w)$ and depth $O(\frac{nk \log n}{w})$.*

Proof of Theorem 4.3.2: In [4], Assaf and Upfal did not address the issue of worst-case faults. However, by following their method, it is possible to construct a k-destructive-fault-tolerant sorting network with width $O(kn)$ and depth $O(\log n)$ when $k = O(\log n)$. So we can use *Column-Sort*, as in the proof of Theorem 4.3.1, to prove the theorem. ■

Chapter 5

Simulation Results on Constructing Small Depth Circuits for Sorting Most Permutations

The problem of constructing small depth sorting circuits has been intensively studied in the past several decades (see [10] for the early history of this problem and [22] for the more recent progress). Despite many efforts, the 25-year-old $\frac{\log n(\log n+1)}{2}$ upper bound on depth discovered by Batcher [5] remains the best known for problem sizes encountered in practice,[1] say, for $n \leq 2^{100}$. In general, constructing small depth sorting circuits has proved to be a hard problem. In fact, testing whether a given circuit is a sorting circuit is co-NP complete (this was first proved by M. Rabin in [8]). As a consequence, proving lower bounds on the depth of sorting circuits is also hard. The best known asymptotic lower bounds and bounds for small n can be found in [25] and [19], respectively.

In this chapter, we present empirical results for constructing small depth circuits that sort most input permutations. The circuits in this chapter are different from those in the other parts of the thesis:

[1]To be more precise, slightly better circuits are known for $n \leq 16$, and Batcher's result can be slightly improved in general by interrupting Batcher's recursive procedure early and substituting Batcher's original construction with the best circuit known for a small value of n. But such a technique can only improve the depth of Batcher's circuit by a small additive constant.

- The circuits in this chapter are not sorting circuits in a rigorous sense in that an n-input circuit of this chapter will only sort most of the $n!$ possible n-input permutations, not necessarily all of them. That is, an n-input circuit in the chapter is allowed to fail on a small fraction of all the $n!$ n-input permutations. We will refer to such circuits as *circuits for sorting most permutations.*[2]

- The functionality of the circuits in this chapter is only empirically tested, and no theoretical analysis has yet been found. Consequently, no theorem will be proved and only simulation results will be presented in this chapter.

Another way of viewing a circuit for sorting most permutations is that the circuit sorts, with a certain high probability, an input permutation that is chosen uniformly at random from the set of all $n!$ permutations (i.e., each permutation is chosen with probability $\frac{1}{n!}$). As a consequence, with high probability, the circuit sorts any n random values chosen uniformly from an arbitrary interval. In practice, such a circuit can be used in the following fashion. Given any input sequence, we first run our circuit on the input sequence. At the end of the circuit, it is fairly easy to check whether the final outputs are sorted or not, by comparing pairs of neighbors. If the outputs are sorted, we stop; otherwise, we scramble the output values and try again. One may argue that the input permutations occurring in practice are unlikely to be chosen uniformly at random. Fortunately, however, our simulation results show that if a nonuniform distribution is assumed and known in advance, we can actually use our method to construct a special-purpose circuit for sorting (with high probability) inputs drawn according to the given distribution. Such special-purpose circuits have depth even smaller than circuits for inputs drawn uniformly. So in the rest of the chapter, we will not address this issue and will focus on constructing circuits for sorting most permutations.

In this chapter, we present an algorithm that automatically generates circuits

[2]The concept of sorting most permutations should not be confused with that of approximate-sorting, defined on page 23. A Δ-approximate-sorting circuit brings every item to within Δ of its correct position on *all* input permutations, whereas a circuit for sorting most permutations *exactly* sorts *most* of the $n!$ permutations.

for sorting most permutations. Empirical evaluation of the circuits thus constructed shows that they sort 98% of the permutations with depth about $4 \log n$ for $n \leq 2^{13}$. These are the first set of circuits that substantially outperform Batcher's circuits for problem sizes encountered in practice, though our circuits are allowed to fail on a small fraction of the input permutations whereas Batcher's circuits sort all permutations. We also construct small depth passive-fault-tolerant circuits that sort most permutations. Empirically, such circuits have depth smaller than the depth of Batcher's circuits, which are not by themselves fault-tolerant.

We do not know if the technique presented in this chapter can be modified to construct small depth sorting circuits, i.e., circuits for sorting *all* permutations. There are reasons for us to suspect that the smallest depth of a sorting circuit may be considerably larger than that for sorting most permutations. Although our techniques are clearly motivated by the theoretical work of Leighton and Plaxton [17], we cannot prove in theory that our technique always leads to good circuits for sorting most permutations, for either small n or large n. Developing a theory that supports our empirical results is an interesting open question.

The remainder of the chapter is divided into two sections. The algorithm for constructing the circuits is presented in Section 5.1, and the empirical results and some figures of the circuits are presented in Section 5.2.

5.1 An Algorithm for Constructing the Circuits

For ease of notation, we will assume throughout the chapter that the registers in a circuit are labeled as $r_1, r_2, \ldots, r_n$, from the top to the bottom.

Before presenting the algorithm for constructing the circuits, and in order to avoid future ambiguity, we first make a clarification. Each of the circuits in the chapter will have the property that there exists a permutation π such that, on most input permutations, the ith smallest item is output to register $r_{\pi(i)}$ for $i = 1, 2, \ldots, n$. In other words, with a fixed permutation π^{-1} applied to the output registers, the circuit sorts most input permutations. In a typical sorting circuit, however, the ith smallest item should be output to the ith register r_i, i.e., permutation π should be the identity permutation. In general, a circuit $\mathcal{C}$ that on input set S outputs the ith smallest item to $r_{\pi(i)}$ can always be converted, by a technique similar to that for solving Exercise 16 on page 239 of [10], into another circuit $\mathcal{C}'$ of the same depth that on input set $\pi'(S)$ outputs the ith smallest item to r_i, where π' is a fixed permutation and $\pi'(S) = \{\pi'(\alpha) \mid \alpha \in S\}$. In other words, we can always force the permutation applied at the end of the circuit to be the identity permutation, without increasing the depth of the circuit.[3] We remark that it is not clear if the technique can be applied to circuits with passive-faults (see the last paragraph in Section 1.3). Hence, the passive-fault-tolerant circuits constructed in Subsection 5.1.5 each has a fixed permutation attached at the end. We assume that such a permutation is hard-wired into the circuit and does not affect the depth. So in the rest of the chapter, we will be content with constructing circuits that, on most input permutations, output the ith smallest item to $r_{\pi(i)}$ for a fixed permutation π.

The remainder of the section is organized into subsections as follows. In Subsection 5.1.1, we give a brief overview of the methodology behind the algorithm. Some important technical details are discussed in Subsections 5.1.2, 5.1.3, and 5.1.4, and pseudocode for the algorithm is given in Subsection 5.1.4. In Subsection 5.1.5, we describe how to modify the algorithm to obtain passive-fault-tolerant circuits for sorting most permutations.

5.1.1 Overview of the Algorithm

This subsection contains a high-level description of the algorithm for constructing small depth circuits for sorting most permutations.

The algorithm has theoretical motivations from the circuits designed by Leighton

[3]This will result in the use of non-standard comparators and a fixed permutation prepended to the beginning of the circuit. Clearly, the fixed permutation at the beginning is not essential to the success rate of the circuit and removal of the fixed permutation will only make the circuit sort a different set (of the same size) of permutations. However, this permutation may not be removed if the circuit is designed for a special set of permutations under a special distribution, which was briefly discussed at the beginning of the chapter.

and Plaxton [17]. Therefore, it is useful to have a little understanding of the Leighton-Plaxton circuits before we present the algorithm.

Consider the following d-round *butterfly tournament* of n players for any $n = 2^d$, where d is a positive integer. Initially, the n players are ordered arbitrarily. In the first round, the player at the ith position plays a match with the player at the $(i + \frac{n}{2})$th position for $i = 1, 2, \ldots, \frac{n}{2}$. In the remaining $d - 1$ rounds, a $(d - 1)$-round butterfly tournament is performed among all the $\frac{n}{2}$ winners, and another $(d - 1)$-round butterfly tournament is performed among all the $\frac{n}{2}$ losers. Clearly, such a tournament is only guaranteed to correctly identify the first and the last (i.e., the best and worst) players. For example, the second player may end up with d possible win-loss records. The novelty of the work by Leighton and Plaxton [17] is their proof that such a tournament has a strong ranking property on average, although not necessarily on all possible initial orders. In particular, if the players are placed in a random order at the beginning, then with high probability, the ranks of most players can be approximated to within n^γ based upon their win-loss records, where γ is a constant strictly less than 1. Such a tournament naturally corresponds to a circuit with matches between players interpreted as comparators between corresponding registers. Using the butterfly tournament in a recursive fashion, Leighton and Plaxton constructed a high-probability sorting circuit with depth $7.44 \log n$.

Unfortunately, the Leighton-Plaxton circuit does not immediately yield a circuit that outperforms Batcher's circuit for problem sizes encountered in practice. First, in order to prove that their circuit has a nontrivial probability of success, they needed to assume that n is sufficiently large. Second, they only provided an existence proof of their circuit and no explicit construction of the circuit was found. In addition, there are reasons to believe that their method does not yield the best possible constant. (They seem to have abandoned some useful information when sorting smaller sets of items in the recursive procedure.)

Despite the mathematical difficulties in the proof, the reason that the butterfly tournament has a strong ranking property can be understood as follows. In the tournament, two players meet in a match only if they have the same record so far.

Therefore, such a match provides a lot of information about the ranks of the players. (The reason that Leighton and Plaxton may have lost some useful information when sorting smaller sets of items is that after the first $\log n$ rounds, each player has a unique record and it is not clear what would be the best strategy to use afterwards.)

Motivated by the strong ranking property of the butterfly tournament, we construct our circuit inductively as follows.

Initial Step. In the first level of the circuit, we arrange $\frac{n}{2}$ disjoint comparators in an arbitrary fashion. For example, we can include comparators between r_i and r_{n+1-i} for $i = 1, 2, \ldots, \frac{n}{2}$. On randomly chosen input permutations, it makes no difference how to arrange the $\frac{n}{2}$ disjoint comparators in the first level since all input permutations are equally likely.

Inductive Step. Assuming that we have constructed the circuit for up to level $l-1$ for some $l > 1$, we next describe how to arrange comparators at the lth level of the circuit. Since we only want a small depth circuit and we do not particularly care about the size of the circuit, we may include as many comparators as we can, under the restriction that each register can only be connected to at most one comparator at level l. So we can assume that each register is connected to exactly one comparator at level l, i.e., no register will be left idle at level l. Moreover, we may assume without loss of generality that each comparator moves its small input value to its top register and its large input value to its bottom register. Now, consider a complete graph, G, with n vertices corresponding to the n registers of the circuit. Thus, each possible comparator at level d of the circuit corresponds to an edge of G, and the set of $\frac{n}{2}$ comparators at level l corresponds to a perfect matching of G. Therefore, the question of determining all the comparators at level l reduces to the problem of determining a certain perfect matching of G. To find an appropriate matching, we will assign a weight to each edge of G. In particular, the weight of the edge between registers r_i and r_j will measure the amount of information obtainable by including a comparator between r_i and r_j at level l. To construct the lth level of the circuit, we will choose a perfect matching of G with an approximately maximum weight under the chosen weight assignment. In the context of ranking n players, our circuit corresponds to a tournament in which we proceed in a greedy fashion by arranging matches among players in order to obtain an approximately maximum amount of information in each round.

The preceding high-level description of our algorithm has not defined either the weight assignment procedure or the method for finding the matching. These two components of the algorithm will be described in the next two subsections.

In theory, when the weight assignment and the method to obtain an approximately maximum weight perfect matching are both given, the resulting circuit is completely determined. In reality, however, accurately computing the weight of a comparator is a very difficult task, because our weight assignment corresponds to a certain probabilistic measure of certain highly dependent random variables. Therefore, instead of accurately computing these probablistic quantities by analytic means, we will estimate their values via simulation techniques. In particular, to approximate the desired probabilistic quantities for constructing the lth level of the circuit, we feed a large number of random permutations into our partial circuit with $l-1$ levels and use the outputs of the partial circuit to estimate the real random variables. In addition to the inaccuracy of the weight computation, as we will see in the next subsection, there are several weight assignments that all lead to good circuits. Given the uncertainties inherent in our choice and calculation of the weights, it seems unnecessary to insist on finding a maximum weight perfect matching in order to construct the lth level of the circuit. (In fact, even if we stick to a fixed weight assignment and if we can compute the weight exactly, it is not clear whether a maximum weight perfect matching will yield the best circuit.) Therefore, we will use a certain heuristic to find an approximately maximum weight perfect matching. The detailed description of the weight assignment and the heuristic for finding the matching are given in the next two subsections.

Since we use a simulation technique to obtain the weight assignment, there are certain uncertainties in the structure of our circuit. For example, the structure of the lth level of the circuit depends on which set of input permutations are used to obtain the weight assignment. However, if we assume the weight can be accurately computed, and if we use a fixed algorithm to obtain the matching, then (for each n) the circuit will be completely determined. (For example, in theory, we can feed in all the $n!$ possible permutations and use a fixed maximum weight perfect matching algorithm.) Even under such assumptions, we do not know how to prove that our technique will lead to a circuit with a reasonable depth and success probability. Developing a theory that supports our empirical results is an interesting open question.

5.1.2 Weight Assignments

Following the notation of the preceding subsection, let G be the complete graph used for constructing the lth level of the circuit. In this subsection, we describe a procedure for assigning a weight to each edge in G. Since each edge of G corresponds to a unique possible comparator at level l, it suffices to define the weight of each possible comparator at level l, which we do in the following.

Let $\mathcal{C}$ denote the partial circuit consisting of the $l-1$ levels constructed so far. Consider an arbitrary pair of registers r_x and r_y in $\mathcal{C}$, where $x < y$. Let $C_{x,y}$ denote the possible comparator between r_x and r_y at level l. When a random input permutation is fed into $\mathcal{C}$, the items contained in r_x and r_y at the end of $\mathcal{C}$ are both random variables. Let X and Y denote these two random variables at r_x and r_y, respectively. If we include a comparator $C_{x,y}$ at the end of $\mathcal{C}$, then the items output to r_x and r_y by $C_{x,y}$ are also random variables. Let us denote these two random variables at r_x or r_y after $C_{x,y}$ by X' and Y', respectively. Intuitively, one would expect the distributions of X' and Y' to be closer to threshold functions than those of X and Y. Our weight assignment to comparator $C_{x,y}$, denoted by $w_\lambda(x,y)$ where λ is a parameter to be chosen later, will measure how much comparator $C_{x,y}$ can bring these distributions closer to threshold functions.

As pointed out in Subsection 5.1.1, $w_\lambda(x,y)$ will be empirically determined. In particular, we feed T randomly chosen permutations to circuit $\mathcal{C}$, with $l-1$ levels. For any i such that $1 \le i \le T$, let x_i be the value of random variable X for the ith trial, i.e., x_i is the item output by $\mathcal{C}$ at register r_x on the ith selected permutation.

Define y_i, x'_i, and y'_i similarly. Then, we define

$$w_\lambda(x,y) = \left(\sum_{1\le i\le T,\ x_i<y_i} (y_i - x_i)^\lambda \right) \left(\sum_{1\le i\le T,\ y_i<x_i} (x_i - y_i)^\lambda \right). \tag{5.1}$$

Note that $w_\lambda(x,y) = 0$, for any λ, if $x_i < y_i$ for $i = 1,2,\ldots,T$ or if $x_i > y_i$ for $i = 1,2,\ldots,T$. This represents the fact that $C_{x,y}$ has little value if the relationship between the items contained in r_x and r_y is already completely determined after the first $l-1$ levels of the circuit.

We now further explain the motivation behind the weight assignment of equation 5.1. First, by definition,

$$w_0(x,y) = \left(\sum_{1\le i\le T,\ x_i<y_i} 1 \right) \left(\sum_{1\le i\le T,\ y_i<x_i} 1 \right) = T^2\, \frac{|\{i : y_i > x_i\}|}{T} \frac{|\{i : y_i < x_i\}|}{T}.$$

When divided by T^2, the preceding expression is clearly an empirical approximation of

$$\text{Prob}(Y > X) \cdot \text{Prob}(Y < X). \tag{5.2}$$

Second, by definition, $x'_i = \min\{x_i, y_i\}$ and $y'_i = \max\{x_i, y_i\}$. Therefore, it is easy to check that

$$y_i - x_i + y'_i - x'_i = \begin{cases} 2(y_i - x_i) & \text{if } y_i > x_i \\ 0 & \text{otherwise,} \end{cases} \tag{5.3}$$

and

$$x_i - y_i + y'_i - x'_i = \begin{cases} 2(x_i - y_i) & \text{if } x_i > y_i \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

Moreover,

$$x'_i + y'_i = x_i + y_i.$$

Therefore,

$$\sum_{1\le i\le T} x'_i + \sum_{1\le i\le T} y'_i = \sum_{1\le i\le T} x_i + \sum_{1\le i\le T} y_i, \tag{5.5}$$

and similarly

$$\sum_{1\le i\le T} x_i'^2 + \sum_{1\le i\le T} y_i'^2 = \sum_{1\le i\le T} x_i^2 + \sum_{1\le i\le T} y_i^2. \tag{5.6}$$

Now, by simple algebraic calculations,

$$\begin{aligned}
& w_1(x,y) \\
&= \left(\textstyle\sum_{1\le i\le T,\, x_i<y_i}(y_i - x_i)\right)\left(\textstyle\sum_{1\le i\le T,\, y_i<x_i}(x_i - y_i)\right) \\
&= \left(\textstyle\sum_{1\le i\le T} \frac{y_i-x_i+y_i'-x_i'}{2}\right)\left(\textstyle\sum_{1\le i\le T} \frac{x_i-y_i+y_i'-x_i'}{2}\right) && \text{(by equations 5.3 and 5.4)} \\
&= \tfrac{1}{4}\left(\textstyle\sum_{1\le i\le T}(y_i'-x_i')\right)^2 - \tfrac{1}{4}\left(\textstyle\sum_{1\le i\le T}(y_i-x_i)\right)^2 \\
&= \tfrac{1}{4}\left(\textstyle\sum_{1\le i\le T}(y_i'+x_i')\right)^2 - \left(\textstyle\sum_{1\le i\le T} y_i'\right)\left(\textstyle\sum_{1\le i\le T} x_i'\right) \\
&\quad -\tfrac{1}{4}\left(\textstyle\sum_{1\le i\le T}(y_i+x_i)\right)^2 + \left(\textstyle\sum_{1\le i\le T} y_i\right)\left(\textstyle\sum_{1\le i\le T} x_i\right) \\
&= \tfrac{1}{2}\left(\textstyle\sum_{1\le i\le T}(y_i'+x_i')\right)^2 - \left(\textstyle\sum_{1\le i\le T} y_i'\right)\left(\textstyle\sum_{1\le i\le T} x_i'\right) \\
&\quad -\tfrac{1}{2}\left(\textstyle\sum_{1\le i\le T}(y_i+x_i)\right)^2 + \left(\textstyle\sum_{1\le i\le T} y_i\right)\left(\textstyle\sum_{1\le i\le T} x_i\right) && \text{(by equation 5.5)} \\
&= \tfrac{1}{2}\left((\textstyle\sum_{1\le i\le T} y_i')^2 + (\sum_{1\le i\le T} x_i')^2 - (\sum_{1\le i\le T} y_i)^2 - (\sum_{1\le i\le T} x_i)^2\right) \\
&= \tfrac{1}{2}\left((\textstyle\sum_{1\le i\le T} y_i')^2 - T\sum_{1\le i\le T} y_i'^2\right) + \tfrac{1}{2}\left((\sum_{1\le i\le T} x_i')^2 - T\sum_{1\le i\le T} x_i'^2\right) \\
&\quad -\tfrac{1}{2}\left((\textstyle\sum_{1\le i\le T} y_i)^2 - T\sum_{1\le i\le T} y_i^2\right) - \tfrac{1}{2}\left((\sum_{1\le i\le T} x_i)^2 - T\sum_{1\le i\le T} x_i^2\right) && \text{(by equation 5.6).}
\end{aligned}$$

When multiplied by $\frac{2}{T^2}$, the preceding right-hand side is clearly the empirical approximation of

$$\mathrm{Var}(Y) + \mathrm{Var}(X) - \mathrm{Var}(Y') - \mathrm{Var}(X'). \tag{5.7}$$

Now, to see why $w_0(x,y)$ and $w_1(x,y)$ are both good measures of the value of comparator $C_{x,y}$, we only need to argue that the quantities in expressions 5.2 and 5.7 are both good measures of $C_{x,y}$. Intuitively, the value of a comparator depends on the amount of information obtainable by including it in the circuit. So the weight of $C_{x,y}$ should be small if $\mathrm{Prob}(x > y)$ is close to 0 or 1, and it should be large if $\mathrm{Prob}(x > y)$ is close to $\frac{1}{2}$. The quantity in expression 5.2 measures how close $\mathrm{Prob}(x < y)$ is to $\frac{1}{2}$, and therefore it is a good indicator of the value of $C_{x,y}$. To see why the quantity in expression 5.7 is also a good indicator for $C_{x,y}$, note that $\mathrm{Var}(X) + \mathrm{Var}(Y)$ is actually independent of comparator $C_{x,y}$ and that $\mathrm{Var}(X') + \mathrm{Var}(Y')$ measures how close the distributions of X' and Y' are to threshold functions.

In fact, our simulations indicate that good circuits can be constructed by using the weight assignment of either w_0 or w_1. It is also interesting to observe that if we use w_0 as the weight assignment and use a maximum weight perfect matching to construct each level of the circuit, then, at least in theory, the first $\log n$ levels of the circuit will be identical to Leighton and Plaxton's butterfly tournament. Intuitively, by comparing items with the same history in the first $\log n$ levels of the butterfly tournament, we ensure that each of the comparators has weight w_0 equal to $\frac{1}{4}$, the largest possible.

However, our best circuits have been obtained by the general weight assignment w_λ with variable λ. In particular, we set λ to be an increasing function of l, the index of the current level to be constructed.

We next explain why increasing λ with l is perhaps a reasonable strategy. For simplicity, let us consider an extreme case where $x_j > y_j$ and $x_i < y_i$ for all $i \neq j$, $1 \leq i \leq T$. Intuitively, the only evidence that $C_{x,y}$ may be a valuable comparator is provided by the jth trial. (Without this evidence, it would look as if $\text{Prob}(X < Y) = 1$ and there would be no reason to compare r_x and r_y.) Now, how should we weight the information provided by the jth trial? The fact that $x_j > y_j$ shows that the behavior of the jth input permutation in the partial circuit $\mathcal{C}$ is highly unusual, relative to the other $T-1$ input permutations. But more information is provided by the jth permutation than the fact that $x_j > y_j$. For example, we may want to pay more attention to a case where $x_j - y_j = n-1$ than to a case where $x_j - y_j = 1$, because the former case is much more unusual and the latter case is somewhat more normal if we want to declare that the item contained in r_x is smaller than that contained in r_y. Intuitively, the more unusual a permutation is, the more attention we need to pay to it. On the other hand, according to equation 5.1, the larger λ is, the more attention we pay to the information provided by unusual permutations. When l is small, we may decide not to pay too much attention to unusual permutations because we still have a chance to work on them in future levels of the circuit. But when l becomes large, we may want to pay more attention to unusual permutations since we have fewer and fewer opportunities to deal with them. This heuristic argument explains why it is reasonable to choose λ as an increasing function of l.

We conclude this subsection with remarks on some other possible methods of weight assignments. As we have pointed out earlier in this subsection, fairly good circuits can be constructed by using w_0. This means that we can use expression 5.2, which measures how close $\text{Prob}(X > Y)$ is to $\frac{1}{2}$. As a natural alternative, we may use

$$\min\{\text{Prob}(X > Y),\ \text{Prob}(X < Y)\}.$$

Our simulation shows that such a weight assignment does lead to good circuits, but circuits thus constructed are not as good as those constructed by using expression 5.2.

As another alternative, the following approach may seem reasonable. To construct the lth level of the circuit, we rank all the registers according to the expected values output to the registers after applying the circuit with $l-1$ levels. Then, at level l, we compare pairs of registers with the nearest ranks. This approach has clear motivation from Leighton and Plaxton's butterfly tournament, and the first $\log n$ levels of a circuit thus constructed are identical to the butterfly tournament, at least in theory. However, such a choice of weight does not yield good circuits for the following reason. For a pair of registers with the nearest ranks, it is possible that the item contained in one of the registers is always larger than that contained in the other register. In such a case, a comparator between these two registers is meaningless.

In expression 5.7, the term $\text{Var}(X)+\text{Var}(Y)$ is actually independent of comparator $C_{x,y}$, and $\text{Var}(X') + \text{Var}(Y')$ is the only term measuring how close the distributions of X' and Y' are to threshold functions. Hence, at least in theory, using either $\text{Var}(X')+\text{Var}(Y')$ or expression 5.7 should lead to the same maximum weight perfect matching. In our simulations, however, we use a certain greedy heuristic for an approximately maximum weight perfect matching, and expression 5.7 turns out to be superior since it tends to prevent the heuristic from running into the undesirable phenomenon described in the preceding paragraph.

5.1.3 A Matching Heuristic

As in the preceding subsection, we use G to denote the weighted complete graph used for constructing the lth level of the circuit. In this subsection, we describe how to find a perfect matching of G with a large total weight.

There are several known algorithms for finding a maximum weight perfect matching, but most of them are very complicated to implement and require a long running time. It turns out that the running time of the matching algorithm will be the bottleneck of the simulation if we use a complicated matching algorithm. This is because we will construct our circuit level by level and we need to solve a matching problem in order to construct each level of the circuit. Moreover, if we insist on finding a maximum weight perfect matching precisely, we need to maintain a large weighted graph. The graph is large enough that we will face a paging problem on a typical workstation (e.g., a SPARC1 with 8Mb memory) for moderately large values of n (e.g., several thousands). In addition to the technical difficulties to implement a maximum weight perfect matching algorithm, as we have pointed out in Subsection 5.1.1, it seems unnecessary to insist on finding a maximum weight perfect matching due to the uncertainties inherent in our choice and calculation of the weights. Therefore, rather than find a maximum weight perfect matching of G precisely, we use the following heuristic to find a perfect matching with a relatively large weight.

At a high level, our heuristic consists of two major phases. The first phase runs in $\frac{n}{2}$ rounds. Within each round, the heaviest available edge is added to the matching. The second phase of the heuristic rematches some edges to improve the total weight of the matching. More precisely, the heuristic proceeds in three steps, as follows.

Step 1. Sort all of the edges of G according to their weights.

Step 2. Form a matching of G in $\frac{n}{2}$ rounds. In each round,

- we first include the heaviest edge in the list of remaining edges into the matching;
- assuming this edge has nodes x and y, we eliminate from consideration all edges having an end point equal to either x or y.

Step 3. Repeat the following procedure until no pair of edges can be found to satisfy inequality 5.8. For each pair of edges (x, y) and (x', y') in the matching, replace (x, y) and (x', y') with (x, x') and (y, y') if

$$w_\lambda(x, y) + w_\lambda(x', y') \leq w_\lambda(x, x') + w_\lambda(y, y'). \tag{5.8}$$

This completes the description of our matching heuristic. It is straightforward to argue that the heuristic eventually terminates by using the fact that each rematching of a pair of edges in Step 3 increases the weight of the matching and there are only a finite number of matchings in G. However, one might be concerned that we may encounter too many rematchings in Step 3 and end up with a huge running time within Step 3. If this were the case, then the advantage of using the heuristic rather than a maximum weight perfect matching algorithm would be lost. In our simulations, we prevent such a bad scenario from occurring by forcing the procedure to stop after a certain number of rounds of rematchings. It appears that such an enforced stop is not crucial in many cases since, as will be discussed in the next subsection, we only consider a relatively small number of edges for the matching in order to avoid the paging problem.

5.1.4 Putting Things Together

Thus far, we have described essentially all of the main features of our algorithm for constructing circuits for sorting most permutations. In this subsection, we further discuss some technical details. Pseudocode for the entire algorithm is given in Figure 5-1.

For simplicity, we continue to use C to denote the $(l-1)$-level circuit constructed so far. Let G be the weighted complete graph for constructing the lth level of the circuit, as described in Subsection 5.1.1, and let M be an $n \times n$ matrix that represents G.

We begin our discussion by touching on space consideration. As we have seen so far, an important step of our algorithm is to find a large weight perfect matching in graph G. Unfortunately, when we run the algorithm on a typical workstation (e.g., a SPARC1 with 8Mb memory), we start having a paging problem to maintain and access the matrix M, as n becomes several thousands. Furthermore, since M is computed by information drawn from a large number of input permutations, our access to M is so frequent that the time spent on the paging problem is too expensive to pay.

To avoid the paging problem, we use the following trick. First, we rank all the registers by $1, 2, \ldots, n$, according to the expected values contained in the registers after application of the circuit $\mathcal{C}$.[4] Then, we pretend that

$$w_\lambda(r_x, r_y) = 0 \tag{5.9}$$

unless

$$|\mathrm{rank}(r_x) - \mathrm{rank}(r_y)| \leq w, \tag{5.10}$$

where w is a parameter that is relatively small compared with n. In other words, we assume that a comparator between r_x and r_y has non-zero weight only if inequality 5.10 is satisfied.

The rationale underlying the simplification made in equation 5.9 and inequality 5.10 is related to a theorem of Leighton and Plaxton [17], which states that the $(\log n)$-round butterfly tournament brings most of the items into a small window around their correct position. According to this theorem, for relatively large l, it is of little interest to compare a pair of registers r_x and r_y at level l unless they satisfy inequality 5.10 for relatively small w. Of course, one might argue that, for small l, we are losing many valuable comparators by confining ourselves to matching registers within small windows. Fortunately, the following observation indicates that for small l, while losing many valuable choices of comparators, the number of possible comparators under investigation is perhaps large enough for us to construct a reasonably good level of the circuit. For simplicity, consider the extreme case where $l = 1$. Since no comparisons have been made before level 1, all of the registers contain random

[4]As assumed in the beginning of this section, each register has a label, such as r_x. The rank of a register is in general different from its label. For example, $\mathrm{rank}[r_x] \neq x$ for most x's.

variables with the same distribution. So the ranks of the registers can be arbitrary, i.e., it is solely dependent on which permutations we happen to choose as our samples and what tie-breaking method we use in the ranking procedure. By eliminating all the pairs of registers that do not satisfy inequality 5.10, we actually eliminate many good choices of comparators. However, this does not mean that we are losing a great amount of information available to us, since the remaining comparators are likely to be as good as those eliminated.

If we make the assumption of equation 5.9 and inequality 5.10, then the only nontrivial entries we need to maintain for matrix M will be those within w positions of the diagonal, up to a fixed permutation based upon the ranks of the registers. This will reduce the amount of memory needed for M from $\frac{(n-1)n}{2}$ to $\frac{w(2n-w-1)}{2}$ (note that M is symmetric and has 0 entries along the diagonal).

Not surprisingly, our simulations show that circuits constructed with large w tend to be better than circuits constructed with small w. Intuitively, we are less restricted when using larger w. Hence, to obtain circuits with good performance, we should set w as large as can afford in terms of computational time and space.

We also need to determine at which level we should stop the algorithm. We can reasonably predict the number of levels needed in an n-input circuit based on our results obtained for constructing circuits with fewer inputs. In our simulation, we construct a circuit with slightly more levels than the number thus predicted, and we use simulation to find the best level at which to end the circuit. Of course, by using more steps, we can always sort a larger percentage of input permutations. This fact is reflected in Table 5.1.

When we come to construct the last few levels of the circuit, for most of the permutations, most of the items are fairly close to their correct positions. But we still want to include a few more levels simply because we want to sort a larger fraction of the set of all $n!$ permutations. On the other hand, since we only obtain information from a random set of input permutations, as opposed to all $n!$ permutations, we have little chance to learn much information about unusual permutations. In fact, towards the end of a circuit, the weight of most comparators is 0 or near 0. Hence, the construction of these levels may be fairly random, due to the use of many 0-weight comparators. One way to get around this problem is to use more permutations, but there is certainly a limit on this approach due to the constraints on our computational power. Instead, we use the following two methods, which both appear to be helpful.

In the first method, near the end of a circuit, we repeatedly compare registers with neighboring ranks as in odd-even transposition sort. That is, for l sufficiently large, we compare the register of rank $2i - 1$ with the register of rank $2i$ for $i \leq \frac{n}{2}$ at levels $\{l + 2j : j \geq 0\}$, and we compare the register of rank $2i$ with the register of rank $2i + 1$ for $i < \frac{n}{2}$ at levels $\{l + 2j + 1 : j \geq 0\}$. (Recall that a register is ranked according to the expected value that it receives.)

In the second method, we construct a level l near the end of the circuit in several rounds. In the first round, based on the weighted graph G, we include some comparators corresponding to a matching with a large total weight, but we do not insist on using a perfect matching. In particular, if matching any unmatched nodes in G would only cause us to include edges with 0 weight, we stop the current round. In the next round, we intentionally remove a constant fraction of the comparators that are randomly chosen within each level preceding level l, and we compute another weighted graph according to the partially damaged circuit.[5] Now, based on the new weighted graph, we can match more registers by using non-zero-weight edges. If all the nodes are matched after this round, we stop; otherwise, we run another round with a larger fraction of comparators removed. We continue in this fashion until all registers are matched.

Our best simulation results are obtained by a combination of both methods. That is, we construct the last few levels by the second method, and we construct the very last few levels by the first method.

Finally, the pseudocode for the entire algorithm is given in Figure 5-1. The parameter w is chosen as large as possible given the space and time constraints on our

[5]We remove the comparators only for the purpose of computing a new weighted graph for level l. When we start constructing level $l + 1$ of the circuit, we start with the circuit where no comparator has been removed. Of course, the final circuit output by the algorithm also has no comparator removed.

Input: n, a positive integral power of 2.

Output: An n-input circuit for sorting most permutations.

```
1. Let $l = 1$.
2. Let $\mathcal{C}$ be the current $(l-1)$-level partial circuit, and let $\mathcal{C}^*$ be the circuit constructed by attaching several levels of an odd-even transposition circuit to the end of $\mathcal{C}$, in the way described in this subsection. Test if $\mathcal{C}^*$ can sort a sufficiently large percentage of permutations. If yes, output $\mathcal{C}^*$ and stop.
3. Rank all the registers in $\mathcal{C}$ by $1, 2, \ldots, n$ according to the expected values contained in the registers after applying $\mathcal{C}$.
4. Construct a complete graph $G$ with $n$ vertices corresponding to the $n$ registers of $\mathcal{C}$. For each edge $(r_x, r_y)$ of $G$, if $|\text{rank}(r_x) - \text{rank}(r_y)| \leq w$, assign a weight to the edge by using equation 5.1; otherwise, assign zero weight to the edge.
5. Find a perfect matching of $G$ by using the heuristic of Subsection 5.1.3. Construct the $l$th level of the circuit according to the matching.
6. Let $l = l + 1$, and go to Step 2.
```

Figure 5-1: An algorithm to construct a circuit for sorting most permutations.

computational power. In practice, this tends to mean that w is chosen sufficiently small to avoid excessive paging and execution time.

5.1.5 Obtaining Passive-Fault-Tolerance

In this subsection, we modify the algorithm described in the previous subsections to obtain an algorithm for constructing passive-fault-tolerant circuits for sorting most permutations. We only consider passive faults in this chapter of the thesis.

In the algorithm for constructing fault-free circuits, we try to add comparators that could yield as much information as possible within each level. In particular, the "information gain" associated with a possible comparator is heuristically defined by equation 5.1. When random faults are introduced, we would still like to follow the same basic strategy. The only change is that we will measure the information obtainable by a possible comparator in a slightly different way. We still use equation 5.1 to compute the weight of a possible comparator at level l, but in order to take into account that some of the comparators will be faulty, we intentionally set a constant fraction of the comparators, chosen at random, to be faulty within each level preceding level l.

To be more precise, suppose we want to construct a circuit that works with high probability even if each gate is faulty with probability upper bounded by a constant ρ_0. In order to assign weights to the edges of the complete graph for level l, we intentionally set $\frac{\rho_0 n}{2}$ randomly chosen comparators to be faulty at each level before and including level $l-1$. Then, we compute the weight of each possible comparator at level l by feeding a large number of input permutations to the faulty $(l-1)$-level circuit and by using equation 5.1. Finally, we find a matching by the heuristic described in Subsection 5.1.3, and construct the lth level accordingly.

The simulation results of two 1024-input circuits thus obtained are shown in Tables 5.2 and 5.3.

5.2 Empirical Results

This section contains our empirical results on circuits for sorting most permutations that are constructed by the algorithms described in Section 5.1.

Table 5.1 contains our simulation results for fault-free circuits that sort most permutations. In this table, d_b denotes the depth of Batcher's circuits, d_n denotes the depth of our new circuits, and p denotes the success rate of our circuits on random permutations. The values of d_n displayed in consecutive columns for the same n are obtained essentially from the same circuit. For example, according to the entries for $n = 1024$ (2^{10}) in the table, we have found a circuit, $\mathcal{C}_1$, of depth 38, and a circuit, $\mathcal{C}_2$, of depth 39. The success rates of these circuits are 93% and 99%, respectively. Actually, $\mathcal{C}_1$ is just a subcircuit of $\mathcal{C}_2$ with the last level removed. In other words, we sort 6% more permutations by including the last level of $\mathcal{C}_2$ than by using $\mathcal{C}_1$ alone.

In our simulations, we have used thousands (or even hundreds, for small values of n) of random "sample" permutations to compute the weights. That is, we choose T to be several thousands in equation 5.1. To make the samples as random as possible, we have used different sets of random permutations for constructing different levels of a circuit. Also, we have tested our circuits on thousands of randomly chosen permutations. The random permutations for testing the success rate of a circuit are generally different from those used during the construction of the circuit. This is very important, because otherwise our circuits might only work well on a small fixed set of permutations, rather than on most permutations.

Observe that our circuits outperform Batcher's circuits starting from the case where $n = 32$ (2^6). Most importantly, for $n \leq 2^{13}$, our circuits sort more than 98% of all permutations with about $4 \log n$ depth.[6] This is the first set of circuits that outperform Batcher's circuits for problem sizes encountered in practice.

$\log n$	6		7		8		9		10		11		12		13		14	
d_b	21		28		36		45		55		66		78		91		105	
d_n	19	20	24	25	28	29	34	35	38	39	44	45	51	52	57	58	62	63
p	.98	.99	.98	.99	.97	.99	.93	.99	.93	.99	.96	.99	.97	.98	.97	.98	.89	.91

Table 5.1: The depth of the new circuits, compared with Batcher's circuits.

Tables 5.2 and 5.3 contain results for two passive-fault-tolerant circuits with 1024 inputs. Table 5.2 shows the results on a circuit constructed with parameter $\rho_0 = 0.02$, and Table 5.3 shows the results for a circuit constructed with parameter $\rho_0 = 0.05$. (The definition of ρ_0 can be found in Subsection 5.1.5.) In these tables, ρ denotes the failure probability of each comparator in the circuit. The parameters in either Table 5.2 or Table 5.3 are for the same circuit. For example, when $\rho = 0$, the circuit of Table 5.2 sorts 95% of all permutations with 42 levels, and it sorts 4% more permutations by using two more levels. Moreover, when $\rho = 0.01$, the same circuit sorts 90% of all permutations with 44 levels, and it sorts 95% of all permutations with 46 levels.

We remark that the success rates given in Tables 5.2 and 5.3 are not obtained by testing a particular randomly faulty version of a circuit on many input permutations,

[6]For $n = 2^{14}$, our circuit needs much more than $4 \log n$ levels to sort 91% of all permutations. Even with slightly more levels, the circuit is unable to sort substantially more permutations. We believe this is because we have to confine ourselves to match registers within a window that is too small (in order to avoid excessive paging), and we expect that it is possible to construct a 2^{14}-input circuit with much smaller depth. In particular, the jump in the depth and the fall in the success rate between $n = 2^{13}$ and $n = 2^{14}$ can perhaps be reduceed by using a larger window size w when more computational power is available.

for such a statistical measure may be too dependent on the particular fault pattern and hence could be very misleading. So instead, the success rates are obtained by running many randomly faulty versions of a circuit on many random input permutations (e.g., in our simulations, we run thousands of faulty circuits on thousands of input permutations). In particular, we test a circuit $\mathcal{C}$ by a large number of instances. In each instance, we feed a random permutation, say, π, into a randomly faulty version of $\mathcal{C}$, say, $\mathcal{C}(F)$. The success rate of $\mathcal{C}$ is the number of $(\mathcal{C}(F), \pi)$ pairs where $\mathcal{C}(F)$ sorts π, divided by the total number of $(\mathcal{C}(F), \pi)$ pairs.

It is interesting to observe that both of the circuits are fairly robust: although the circuits are constructed with parameter ρ_0, they perform very well for parameter $\rho < \rho_0$. For example, with no faults, they sort a vast majority of all permutations. Most importantly, the circuits have depth smaller than Batcher's circuits, which are not by themselves fault-tolerant.

ρ	0		0.01		0.02	
depth	42	44	44	46	50	52
success rate	.95	.99	.90	.95	.91	.94

Table 5.2: A 2^{10}-input circuit constructed with $\rho_0 = 0.02$.

ρ	0		0.02		0.05	
depth	48	49	51	52	58	61
success rate	.98	.99	.95	.98	.85	.88

Table 5.3: A 2^{10}-input circuit constructed with $\rho_0 = 0.05$.

Finally, we include a few computer-generated figures of our circuit and Batcher's bitonic circuit with 64 inputs. In these figures, sloping lines are used to represent registers. This should be compared with Figure 1-1, where registers are drawn by straight lines only. We use the sloping lines to make sure that the ith smallest item is output to the ith output register. Also, we draw a comparator in a way that is totally different from Figure 1-1. In particular, we use a pair of crossing lines to represent a comparator. We choose not to use a vertical line to represent a comparator (such as in Figure 1-1) since many comparators would overlap each other if we chose to draw comparators by vertical lines when generating a figure on a computer. For example, two possible comparators are shown in Figure 5-2.

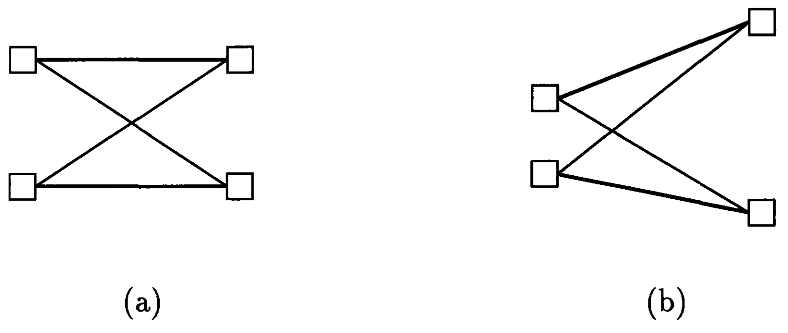

Figure 5-2: Each of the above figures depicts a comparator. In each of the figures, the two bold lines represent the registers, and the two remaining lines represent the comparator.

Figure 5-3 shows a 64-input circuit with depth 19 that sort 98% of all permutations, as claimed in Table 5.1. The same circuit is shown again in Figure 5-4, in which each comparator is shaded according to the frequency that it swaps its two input values: the more frequently a comparator swaps its input values on random permutations, the darker it appears in the picture.[7] Such a shaded picture gives us good understanding of the functionality of the circuit on random permutations. For example, towards the end of the circuit, more and more comparators appear to be lightly shaded or disappear completely in the shaded picture. This corresponds to the fact that near the end of the circuit, more and more permutations are sorted and more and more comparators are left idle or nearly idle.

Lastly, for comparison with Figures 5-3 and 5-4 of our circuit, Batcher's bitonic circuit with 64 inputs is shown in Figure 5-5 and 5-6.

[7]The degree of shading of a comparator is not exactly proportional to the frequency that it swaps its input values. If we were to draw the picture in such a way that the degree of shading were exactly proportional to the frequency, then some comparators would not be visible if they only swap their input values once or twice on the set of randomly chosen set of input permutations. Instead, we have chosen to show all comparators that are not idle on the entire set of randomly chosen permutations.

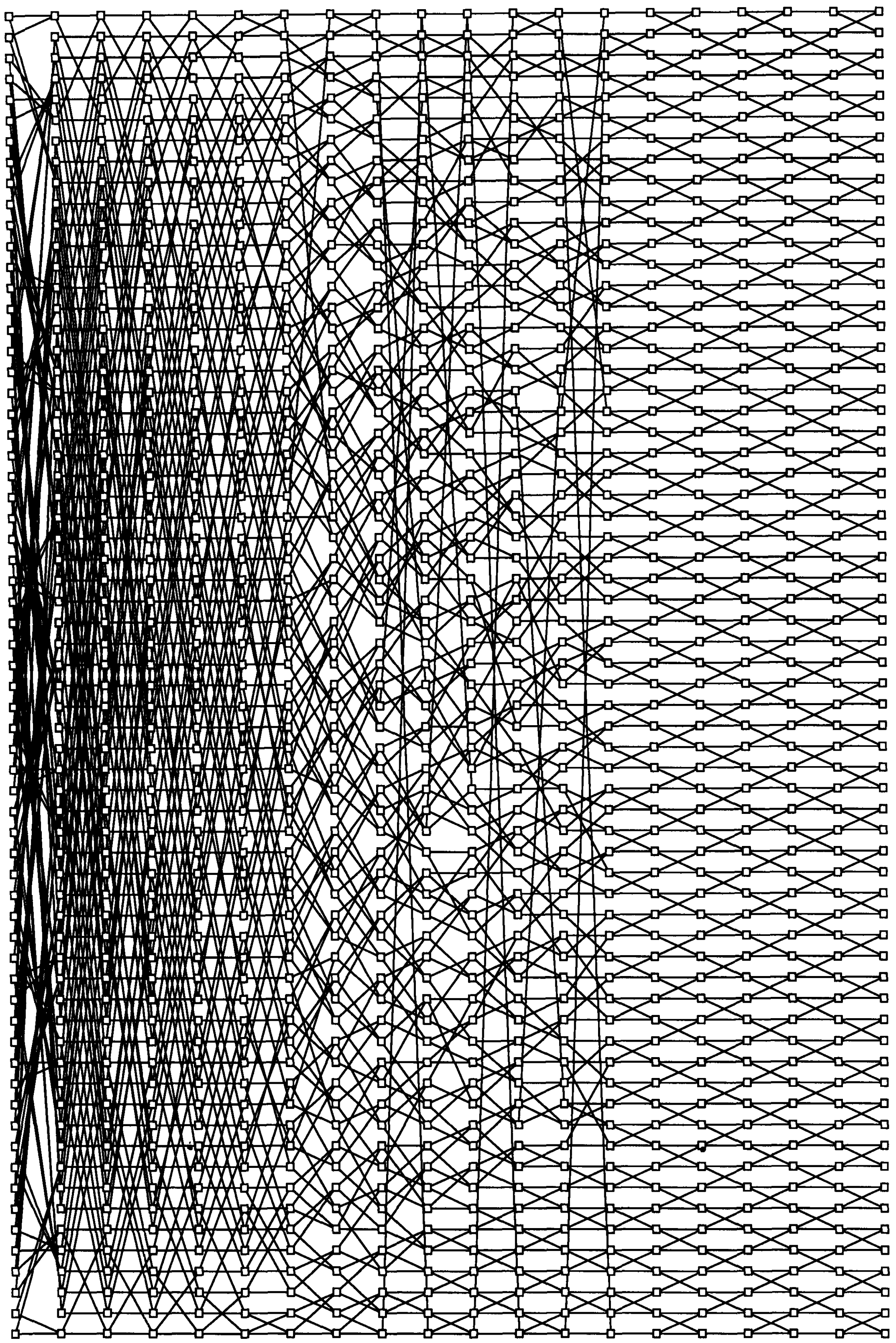

Figure 5-3: A picture of our circuit with 64 inputs.

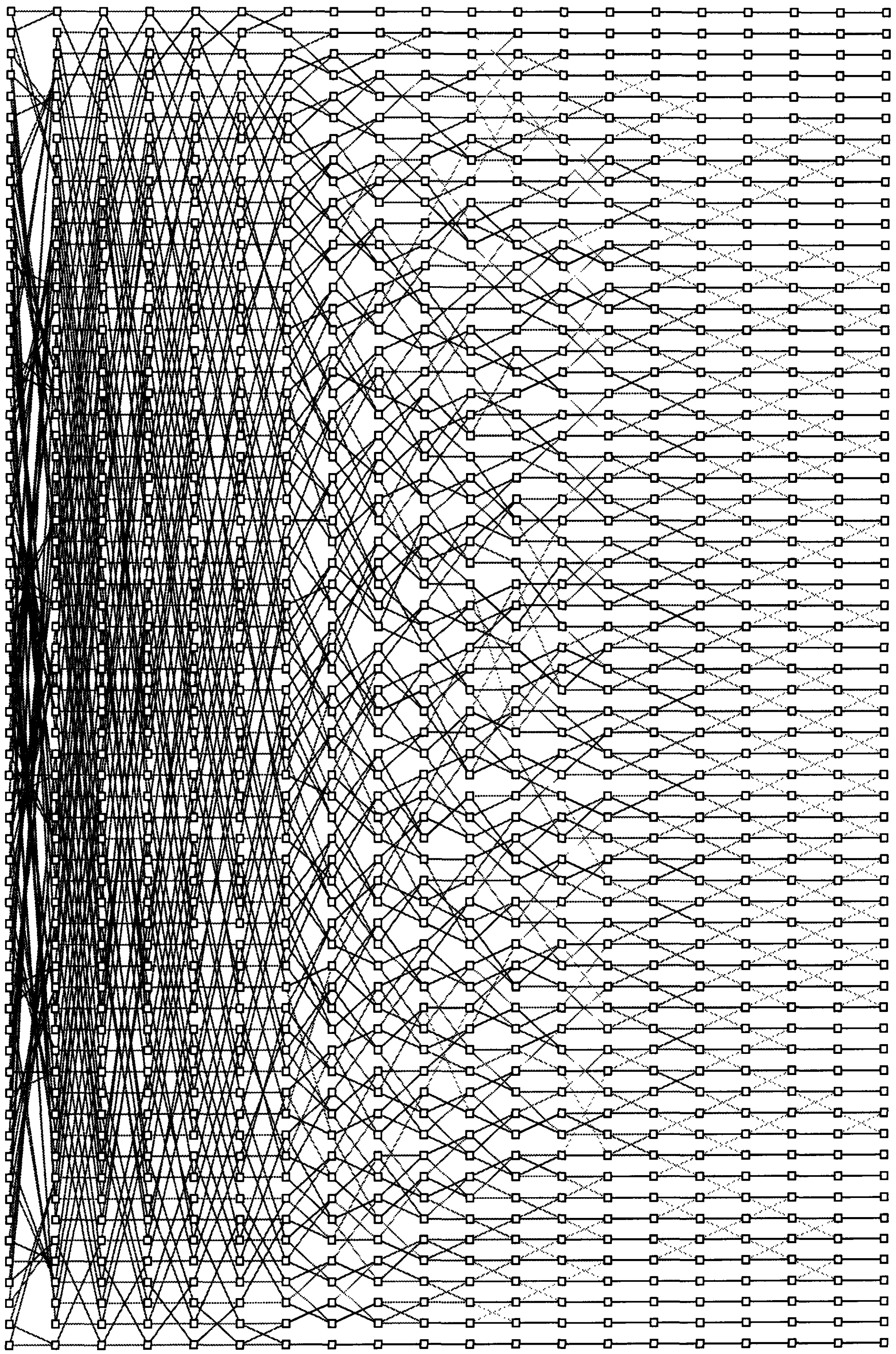

Figure 5-4: A shaded picture of our circuit with 64 inputs.

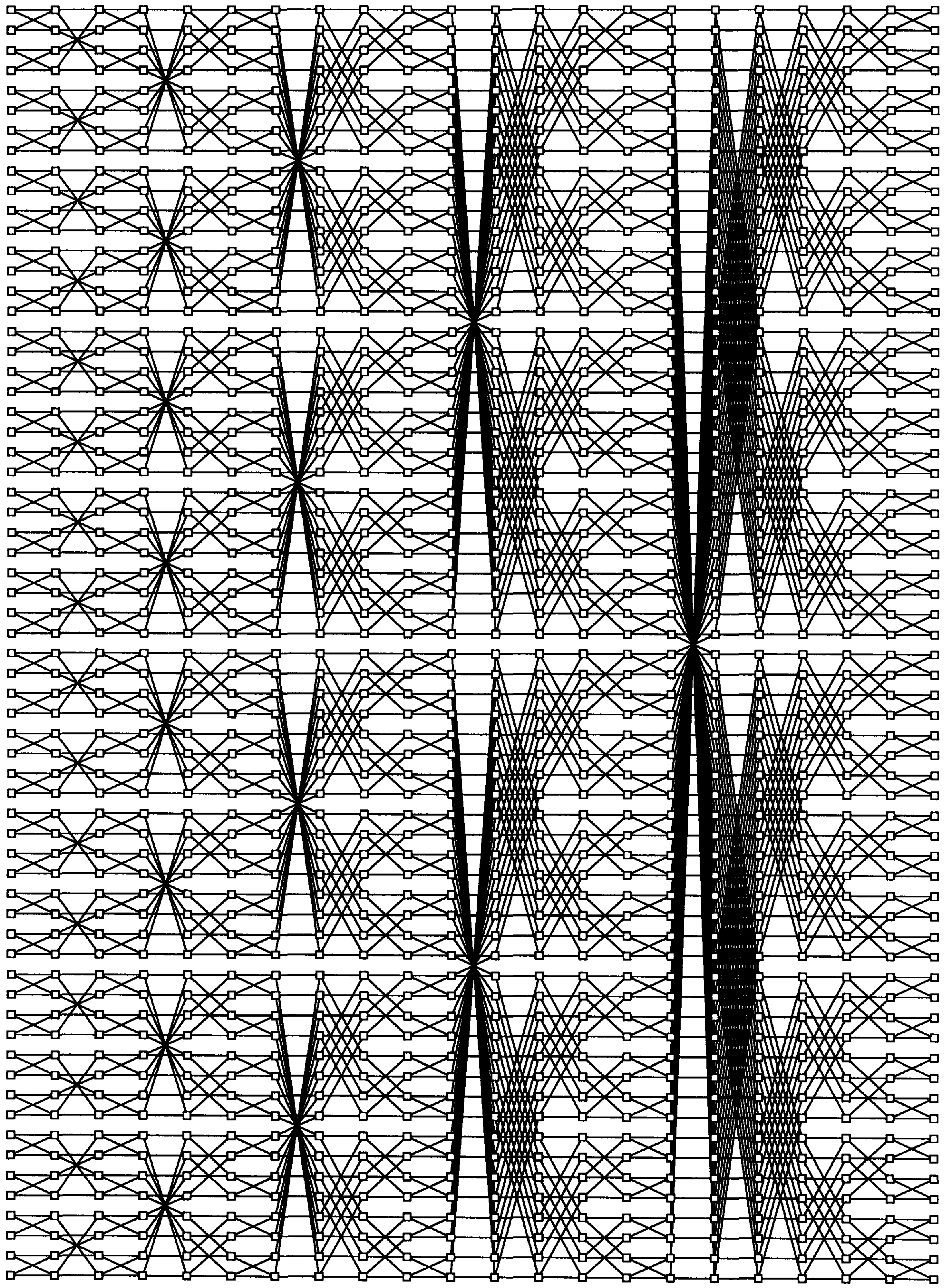

Figure 5-5: A picture of Batcher's bitonic circuit with 64 inputs.

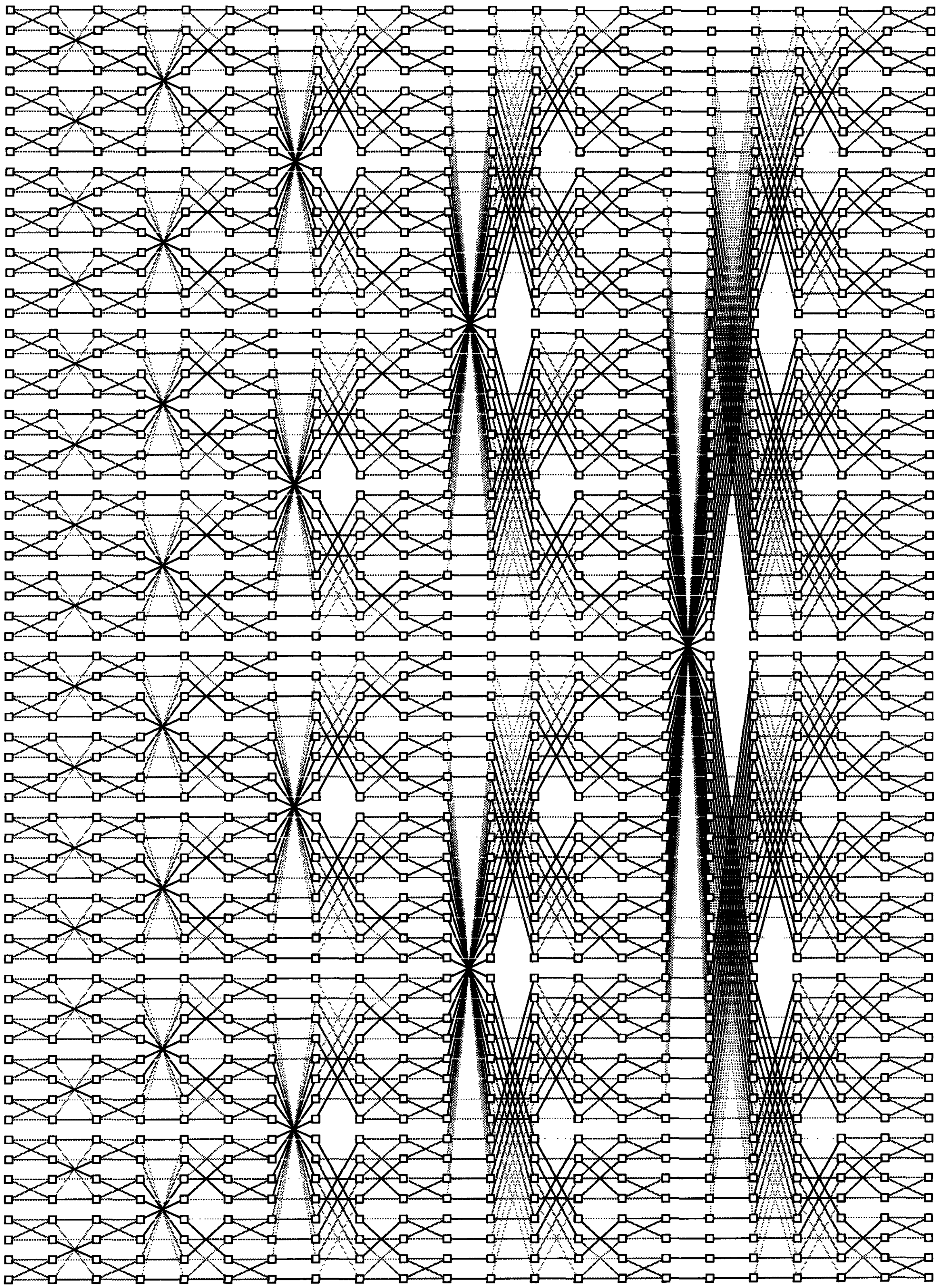

Figure 5-6: A shaded picture of Batcher's bitonic circuit with 64 inputs.

Chapter 6

Conclusions

In this thesis, we have designed various types of fault-tolerant sorting circuits, networks, and EREW PRAM algorithms that break the $\Theta(n \log^2 n)$ barrier on the number of comparators or comparisons. In particular, our EREW PRAM algorithms have achieved the asymptotically optimal running time on a linear number of processors. Also, we have proved a tight lower bound of $\Omega(n \log^2 n)$ on the size of destructive-fault-tolerant sorting networks. All of the upper bound results are based on a new analysis of the AKS circuit, which is of independent interest.

Since substantial progress has been made on the upper bounds on the size of passive-fault-tolerant sorting circuits and reversal-fault-tolerant sorting networks, it would be of great interest to prove some nontrivial lower bounds on the size of passive-fault-tolerant-sorting circuits and reversal-fault-tolerant sorting networks.

For random passive faults, Yao and Yao [26] conjectured in 1985 that $\omega(n \log n)$ size is required even for merging circuits. While to prove (or disprove) Yao and Yao's long-standing conjecture still seems to be fairly hard, as an intermediate step, it would be interesting to prove a $\omega(\log n)$ lower bound on the depth of passive-fault-tolerant sorting circuits. In particular, we do not know if the much simpler problem of insertion can be solved with an $O(\log n)$ depth circuit.

We conjecture that reversal-fault-tolerant sorting networks must have $\omega(n \log n)$ size. Some of the techniques for attacking this problem may have been developed in [9]. Moreover, in all our upper bound results for random reversal faults, we have assumed the failure probability of each comparator to be sufficiently small. It would be interesting to know if these upper bounds still hold when the failure probability of each comparator is near $\frac{1}{2}$.

Finally, we have presented simulation results for constructing small depth circuits that, either with or without faults, sort most permutations. Our circuits are the first that outperform Batcher's circuits for problem sizes encountered in practice. It would be very interesting to develop a theory to support our simulation results.

Bibliography

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on the Theory of Computing*, pages 1–9, May 1983.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[3] V. E. Alekseyev. Sorting algorithms with minimum memory. *Kibernetika*, 5:99–103, 1969.

[4] S. Assaf and E. Upfal. Fault-tolerant sorting network. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 275–284, October 1990.

[5] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 32, pages 307–314, 1968.

[6] K. Diks, A. Pelc, and M. Piotrow. Fault-tolerant networks for fast parallel minimum finding. *Research Manuscript*, 1993.

[7] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Computing with unreliable information. In *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, pages 128–137, May 1990.

[8] D. Johnson. NP-completeness column. *J. Algorithms*, 3:298, 1982.

[9] D. Kleitman, T. Leighton, and Y. Ma. On the design of reliable Boolean circuits that contain partially unreliable gates. *Research Manuscript*, 1994.

[10] D. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, 1973.

[11] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes.* Morgan-Kaufmann, San Mateo, CA, 1992.

[12] T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34:326–335, 1985.

[13] T. Leighton and Y. Ma. Breaking the $\Theta(n \log^2 n)$ barrier for sorting with faults. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 734–743, November 1993.

[14] T. Leighton and Y. Ma. Tight bounds on the size of fault-tolerant merging and sorting networks with destructive faults. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 30–42, June 1993.

[15] T. Leighton, Y. Ma, and G. Plaxton. Highly fault-tolerant sorting circuits. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 458–469, October 1991.

[16] T. Leighton and B. Maggs. Expanders might be practical: Fast algorithms for routing around faults in multibutterflies. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 458–469, October 1990.

[17] T. Leighton and G. Plaxton. A (fairly) simple circuit that (usually) sorts. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 458–469, October 1990.

[18] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

[19] I. Parberry. A computer-assisted optimal depth lower bound for nine-input sorting networks. *Mathematical Systems Theory*, 24:101–116, 1991.

[20] M. S. Paterson. Improved sorting networks with $O(\log N)$ depth. *Algorithmica*, 5:75–92, 1990.

[21] N. Pippenger. On networks of noisy gates. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 30–36, October 1985.

[22] N. Pippenger. Communication networks. In *Handbook of Theoretical Computer Science, J. van Leeuwen, ed., North-Holland, Amsterdam*, 1990.

[23] L. Rudolph. A robust sorting network. *IEEE Transactions on Computers*, 34:344–354, 1985.

[24] M. Schimmler and C. Starke. A correction network for N-sorters. *SIAM J. Comput.*, 18:1179–1187, 1989.

[25] A. C. Yao. Bounds on selection networks. *SIAM J. Comput.*, 9:566–582, 1980.

[26] A. C. Yao and F. F. Yao. On fault-tolerant networks for sorting. *SIAM J. Comput.*, 14:120–128, 1985.

[illegible]